UNIVERSITETI POLITEKNIK I TIRANËS
FAKULTETI I ARKITEKTURËS DHE URBANISTIKËS
DEPARTAMENTI I ARKITEKTURËS

DISERTACION

PËR FITIMIN E GRADËS SHKENCORE "DOKTOR"

FORMA URBANE
dhe
CIKLI I RIPËRTËRITJES ADAPTIVE

Kandidati:

Msc. Renilda HYSENI

Udhëheqës Shkencor:

Prof. Dr. Florian NEPRAVISHTA

Tiranë, 2022

Tezë doktorate

Universiteti Politeknik i Tiranës

Fakulteti i Arkitekturës dhe Urbanistikës

Departamenti i Arkitekturës

Titulli:

FORMA URBANE DHE CIKLI I RIPËRTËRITJES ADAPTIVE

Kandidati:

Msc. Renilda HYSENI

Udhëheqës Shkencor:

Prof. Dr. Florian NEPRAVISHTA

Juria e miratuar:

Prof. Dr. Andrea Maliqari	Kryetar
Prof. Dr. Armand Vokshi	Anëtar
Prof. Asoc. Dr. Denada Veizaj	Anëtar/ Oponent
Prof. Dr. Sokol Dervishi	Anëtar
Prof. Dr. Vlora Navakazi	Anëtar / Oponent

Tiranë, 2022

DEKLARATË

Unë e nënshkruara **Renilda Hyseni**, kandidat doktorant pranë Fakultetit të Arkitekturës dhe Urbanistikës, të Universitetit Politeknik të Tiranës, deklaroj me përgjegjësi të plotë se ky material është puna ime individuale, e pa kopjuar nga puna e askujt tjetër (e publikuar ose jo). Unë konfirmoj njohjen e Rregullore e Studimeve të Doktoratës të UPT, Rregullore e Zhvillimit Kurrikular dhe Mësimdhënies të UPT, Kodin e Etikës si dhe masat e parashikuara në to për plagjiaturën.

Renilda Hyseni

MIRËNJOHJE

Falenderoj profesorin udhëheqës Florian Nepravishta për ndihmën e pakursyer, gjithë stafin akademik dhe mbështetës të Fakultetit të Arkitekturës dhe Urbanistikës.

Simulimet kompjuterike nuk do të ishin realizuar pa punën e palodhur të Gentian Lloshit, i cili mundësoi përpunimin dhe nxjerrjen e të dhënave nga *OpenStreetMap* dhe programimin në *Python*.

Dedikuar familjes sime, në veçanti fëmijëve të mi Sofi dhe Greg, të cilët kanë qenë burim energjie dhe motivimi për përmbylljen me sukses të këtij disertacioni.

PËRMBLEDHJE

Qytetet e zhvillojnë formën e tyre në kohë dhe në hapësirë. Më tepër se hapësira inxhinierike të projektuara, qytetet janë hapësira njerëzore dhe studimi i formës së qytetit, zbulon funksionin antropologjik të tij, duke qendërzuar qeniet njerëzore si krijesa të gjalla me aktivitetin e tyre. Qytetet janë një shembull i sistemeve komplekse adaptive që konfigurohen fizikisht si përmes ndërhyrjeve planifikuese nga lart-poshtë ashtu dhe nëpërmjet decentralizimit, proceseve vetë organizuese nga poshtë-lart. Disa forca të brendshme ndikojnë në strukturën ose rritjen e sistemit dhe përbërësit e tij rrisin koherencën e brendshme të strukturës së tij për shkak të atij vet. Organizimi i shoqërisë varet nga mënyra si njerëzit ndërveprojnë në përditshmërinë e tyre. Ky ndërveprim, në masën më të madhe orientohet nga rrjetet rrugore (Lynch, 1960).

Në përpjekje për të lexuar drejt formën urbane, ky studim vendos në komunikim disa fusha teorike që konsiderohen të përshtatshme për adresimin e këtij kërkimi. Pas prezantimit teorik të formës urbane, nis zgjerimi i konceptit të kompleksitetit dhe ripërtëritjes adaptive. Kompleksiteti i formës urbane mund të analizohet në disa aspekte si: kompleksiteti vizual, kohor, hapësinor dhe strukturor. Ky studim përqendrohet në matje të kompleksitetit strukturor të rrjeteve, nëpërmjet soft-it *OSMnx*, i cili bazohet në teorinë e grafeve. Qëllimi i këtij studimi është hulumtimi i formës urbane në ciklin e ripërtëritjes adaptive në nivel qyteti, kampionësh urban dhe unifikimin e karakteristikave të përbashkëta, me qëllim implementimin në fushën e planifikimit.

Nëpërmjet matjeve metrike dhe strukturore të rrjetit (densitet, lidhshmëri dhe qendërsi), në nivel qyteti dhe në nivel kampionësh, mund të krijojmë një panoramë më të plotë të natyrës së formës urbane të këtyre qyteteve, çka vërteton dhe hipotezën e parë të këtij studimi. Densiteti dhe lidhshmëria ofrojnë kontribut në leximin e formës në të dyja shkallët e observimit, si në nivel qyteti ashtu dhe në nivel kampionësh. Ndërsa qendërsia mund të vëzhgohet vetëm në nivel qyteti. Në këtë aspekt, mund ti përgjigjemi pyetjes së kërkimit, që produkti i ndërhyrjeve nga poshtë-lart, për zhvillimet adaptive paraqet një strukturë më të qëndrueshme, sesa ato të planifikuara. Gjithashtu, kampionë me gjeneza të njëjta, në qytete të ndryshme ofrojnë vlera të njëjta në aspektin e dendësisë dhe lidhshmërisë së rrjeteve. Nga tre tipologjitë e kampionëve të marrë në studim, indi vernakular shfaqet në vlera më të qëndrueshme si në aspektin e dendësisë ashtu edhe në atë të lidhshmërisë.

Në përfundim, ky studim ka një qasje të re në kontekstin shqiptar, leximin e formës urbane përmes rrjeteve. Metodologjia dhe rezultatet e këtij studimi, mund hapin këndvështrime të reja, me qëllim përmirësimin në nivel aplikativ në fushën e planifikimit.

Fjalë kyçe: sisteme komplekse, rrjetet, forma urbane, ripërtëritje adaptive, lidhshmëri

ABSTRACT

Cities develop their form over time and space. Rather than engineered-designed spaces, cities are human spaces, and the study of the city's form reveals its anthropological function, centering human beings as living creatures with their own activity.

Cities are a good example of adaptive complex systems that are physically arranged either by top-down urban interventions or decentralization, as well as bottom-up, self-organized processes. There are internal forces that affect the structure and the system's growth. Some internal forces affect the structure or growth of the system, and its components increase the internal coherence of its structure because of itself. The organization of society depends on how people interact in their everyday lives. This interaction is largely driven by road networks (Lynch, 1960).

With the intention of wading through urban form, this study discusses a couple of theoretical frameworks that are considered appropriate for addressing this research. After introducing the urban form theoritically, the concept of complexity and adaptive resilence is widely developed. The complexity of urban form may be analyzed in some aspects, such as: visual, spatial, structural, and time complexity. This study focuses on measuring the structural complexity of networks through OSMnx software, which is based on graph theory. This study aims to explore the urban form in the cycle of adaptive resilence at the city level, urban samples, and the unification of common characteristics, with the purpose of implementation in the field of urban planning.

The first hypothesis of this study is supported by the network's metric and structural measurements (density, connectedness, and centrality), both at the city and sample levels. These measurements allow us to build a more comprehensive picture of the nature of the urban form of these cities. Density and connectivity provide a contribution to the reading of form at both scales of observation, both at the city level and at the sample level. While centrality can only be observed at the city level. In this regard, we can answer the research question by stating that adaptive developments produced by bottom-up interventions typically have more stable structures than the planned ones. Moreover, samples with the same origins, in different cities, offer the same values in terms of density and connectivity of networks. The vernacular tissue manifests in more stable values from the three typologies of the samples included in the study, both in terms of density and connectivity.

In conclusion, this study has a new approach in the Albanian context, reading the urban form through networks. The methodology and results of this study can open new perspectives, in order to improve urban planning in terms of implementation.

Keywords: complex systems, network, urban form, adaptive resilience, connectivity

Tabela e përmbajtjes:

BAZAT E KËRKIMIT 1

1. BAZAT E KËRKIMIT

1.1 Sfondi i përgjithshëm

"Moria e formave është e pafundme: deri sa çdo formë ka gjetur qytetin e saj, qytete të rinj do të vazhdojnë të lindin. Fundi i qytetit fillon kur format shterojnë larminë e tyre dhe shpërbehen pa shkak."

(Calvino & Weaver, 1974)

Qytetet historike, në kohëra kanë treguar se kanë pasur kapacitetin të përthithin transformime të suksesshme pa humbur strukturën e tyre. Parisi, i ashtuquajturi kryeqyteti i shekullit të XIX-të nga Walter Benjamin (1937), ka mbijetuar në kufijtë e tij historik dhe ende qyteti e ruan karakterin e tij, falë strukturës të krijuar nga Baron Haussman. Në historikun e qyteteve evropiane, karakteri kompleks dhe rrjeti rrugor, i kanë gjurmët që në mesjetë, ndonjëherë edhe në perandorinë romake. Aftësia e qytetit të ruajë identitetin e tij pavarësisht ndryshimeve, shuhet në qytetet moderne, derisa ai humb karakterin e tij dallues dhe fuqinë e tij transformuese.

Aftësia për tu mbijetuar fatkeqësive natyrore dhe të ringrihet edhe pas hirit, si Lisbona pas tërmetit të 1755, Londra pas zjarrit të madh të 1666, Tokio pas tërmetit të 1923, është ajo çfarë ne e quajmë "*urban resilience*[1]*"*, një koncept shumë kompleks i lidhur me qëndrueshmërinë e një kujtese sociale, simbolike dhe materiale. Pjesa më e madhe e qyteteve historike janë "*resilient*[2]" dhe kanë arritur të mbijetojnë për shekuj shpesh duke tejkaluar qytetërimet që i krijuan ato. Diskutimi ngrihet, nëse qytetet moderne mund të mbijetojnë dhe a mund ato të përballojnë testin e kohës si Roma dhe shumë qytete të tjera të krijuara nga perandoria romake rreth Mesdheut? Në kohën kur rreziqet rriten dhe ndryshimet klimaterike sa vijnë e bëhen më të ndjeshme, si parashikohet ekzistenca e tyre?

Qytetet historike janë një standard ideal por jo një model i pastër për tu përsëritur, duke u shfaqur si organizma në ndryshim, dhe të gjitha të ndryshme. Qytetet me origjinë romake kanë cilësi të veçanta dhe elemente që rrjedhin nga principet më të zakonshme sesa një plan formal. Me kohë qytetet rriten dhe bëhen më komplekse në mënyrën e tyre duke përfshirë memoriet e ndërgjegjshme dhe të pandërgjegjshme. Ti heqësh një qyteti memorien e tij, është ti shkatërrosh identitetin dhe tiparet e tij të veçanta, të copëtosh linjat dalluese të zhvillimit të zhdukësh identitetin dhe vlerat e tij. "*Qyteti i Firences është një realitet konkret*" ka shkruar Aldo Rossi (1986). *"Por memoria e Firences dhe imazhi i saj janë të mbushura me vlera që reflektojnë*

[1] aftësi ripërtëritëse
[2] karakteri ripërtëritës

eksperiencën e të tjerëve. Për më tepër, vlerat universale të eksperiencës së krijuar kurrë nuk mund të shpjegojë plotësisht atë gjënë e veçantë që e bën Firencen, Firence."

Nga zëvendësimi i morfogjenezës organike të qyteteve me planet normative të projektuara në hapësira të çliruara nga pesha e kulturës dhe historisë, modernizmi i Le Corbusier zëvendësoi shumëllojshmërinë e pafundme të botës njerëzore me karakterin mekanik të prodhimit në seri (Salat, Labbé, Nowacki, & Walker, 2011). Koha, përkohshmëria dhe kohëzgjatja, të gjitha kanë një ndikim vendimtar, por nuk duhet të harrojmë vlerat e një ritmi të ngadaltë. Informacioni afatgjatë dhe gradual i shpërndarjes në një botë të copëtuar krijoi shumëllojshmërinë e qyteteve perëndimore. Një mantel i larmishëm veshi Mesdheun në të dy anët e tij, në një kohë që Kina e centralizuar zhvilloi një sistem urban më homogjen. Në ndryshim nga kompleksiteti dhe shumëllojshmëria evropiane, struktura urbane e qyteteve amerikane, të cilat janë planifikuar në një kohë kur informacioni përhapej shumë shpejt, janë më homogjene, të projektuara me rrjete ortogonale e të pranishme kudo. Qytetet nuk mund të zgjedhin uniformitetin apo larmishmërinë si çështje zgjedhjesh në planifikimin urban; ato janë produkt i centralizimit politik përballë fragmentizimit, dhe kulturës homogjene përballë diversitetit.

Sidoqoftë, nuk ka rëndësi se sa forma të ndryshme një qytet kalon, faza fillestare e krijimit të tij do të jetë atributi më identifikues për morfologjinë e tij. Kombet dhe shtetet ikin, ndërkohë që qytetet ngelen. "*Shpejt ne do të harrojmë botën dhe shpejt bota do të harrojë ty*", shkruan Marcus Aurelius, por Roma e tij është akoma *"Roma e Fellinit".* Serge Salat (2011), përshkruan se qytetet kanë ndryshuar më shumë në 20 vitet e fundit sesa kanë ndryshuar në dy mijë vitet e fundit. Ndër të tjera ai përmend, se evoluimi i tyre kalon në disa faza, si: *(i) aftësia për të pasur funksione të ndryshme në të njëjtën formë, ose për të ndryshuar formën gradualisht, është adaptimi që karakterizon qytetet historike; (ii) Edhe pse qyteti ndryshon me kohë, tipare të veçanta fizike si rrjeti rrugor dhe struktura urbane mbeten të pandryshuara; (iii) Natyra gjithëpërfshirëse e qyteteve nuk lidhet vetëm me formën dhe funksionin, por edhe me një element të tretë që janë rrjetet;*

Rrjetet janë pa dyshim elementi më thelbësor për të krijuar një qytet të jetueshëm dhe të qëndrueshëm. Diskutimet mbi formën urbane dhe rrjetet rrugore në një qytet, kanë qenë gjithmonë të ndërlidhura me termin e kompleksitetit, dhe e kanë zanafillën që nga Jane Jacobs (1961), e cila e definoi qytetin si një "*kompleksitet të organizuar*", e deri tek Christofer Alexander (1965), e Stephen Marshall (2012) të cilët përcaktuan që *"qyteti nuk është një pemë".* Ata patën qasjen që projektuesit duhet të ofrojnë atë që i mungon qytetit, duke maksimizuar diversitetin në konceptin e teorive të kompleksitetit. Më tej, Emily Talen (2011), kishte qasjen e rregullimit të formës urbane duke balancuar zhvillimet në ndryshim të vetë organizuara nga poshtë-lart, me iniciativat nga lart-poshtë, nëpërmjet vendimmarrjeve të posaçme. Marshall (2012) pohon se ndërhyrje të tilla, janë artificiale pasi zhvillimet për të mirën publike anashkalojnë përzgjedhjen individuale. Në fakt, këto ndërhyrje mund të nxjerrin nga ekuilibri një ekosistem që është krijuar përmes organizimit të decentralizuar nga poshtë-lart.

Në këtë kërkim, synohet të zhvillohet një metodë për të vlerësuar formën urbane dhe kompleksitetin e saj, veçanërisht përmes rrjeteve urbane. Në një qytet rrjetet evoluojnë si

përmes vetorganizimit në kohë, ashtu edhe nëpërmjet proceseve vendimmarrëse planifikuese, por gjithmonë ato influencojnë ndërveprimet njerëzore brenda qytetit. Pra ky studim, zhvillohet në konceptin teorik midis formës urbane, rrjeteve të rrugëve dhe studime të kompleksitetit në funksion të tyre. Nëpërmjet realizimit të matjeve sasiore të rrjeteve, bëhet analiza e tyre, duke u fokusuar në dendësinë, lidhshmërinë dhe qendërsinë.

1.1.1 Konteksti lokal

Zhvillimet urbane në Shqipëri reflektojnë në hapësirë transformimin historik, social-ekonomik, nëpër të cilin ka kaluar shoqëria shqiptare. Pas viteve ’90, ky transformim kryesisht është karakterizuar nga një energji më e madhe e individit dhe më pak e shtetit orientimin e saj. Parë në retrospektivë, zhvillimi urban i qyteteve shqiptare reflekton shtresëzime morfologjike, me origjina të ndryshme. Në qendrat e qyteteve tona, mund të evidentosh gjurmë të arkitekturës mesdhetare (vernakulare), indin otoman, ndikimin italian të viteve ‘30, urbanizimin komunist, zhvillimet informale të pas viteve ’90 dhe zhvillime të orientuara përmes vendimmarrjeve të viteve të fundit. Kështu këto mbivendosje, shpesh janë kontradiktore dhe prodhojnë kompleksitet në leximin e formës urbane.

Në kontekstin shqiptar ka pak studime që lexojnë formën urbane të qyteteve shqiptare në nivel rrjetesh. Në disertacionin e Denada Veizaj (2016), është ofruar një metodë sasiore e bazuar në tre indekse fraktale që përshkruan në terma identitare hapësinore gjurmën fizike të një indi urban specifik. Ky studim, ka bashkuar koncepte të gjeometrisë fraktale me teorinë e rrjeteve dhe midis të tjerave ka arritur në përfundimin se: *“shkalla e lartë e fragmentimit, mund të përkufizohet si një kombinim i një vlere të ulët të dimensionit fraktal, me një indeks të ulët lakunariteti dhe me indeks te lartë dentriciteti. Shkalla e ulët e fragmentimit mund të përkufizohet si një kombinim i një vlere të lartë dimensioni fraktal, vlere të lartë lakunariteti dhe një vlere të ulët dentriciteti.”* Gjithashtu, kërkues tjetër shqiptar Ermal Shpuza (2021), ka zhvilluar kërkime lidhur me transformimin e formës urbane në Shkodër gjatë periudhës otomane, duke u bazuar në teoritë e sintaksës hapësinore. Ai ka arritur në përfundimin se “*gjurmët e rrjeteve të periudhës otomane në Shkodër janë të ngjashme me ato të qyteteve të tjera ballkanike*.”

Ky do jetë i pari studim, i cili krahason formën urbane, parë në optikën e rrjeteve, në qytetet shqiptare me morfogjeneza të ndryshme. Si qytete, janë përzgjedhur qytetet Tirana (Fig. 1, si një metropol, me rrjet të dendur dhe mbivendosje formash urbane), Fieri (Fig. 2, qytet i zhvilluar kryesisht gjatë periudhës socialiste) dhe Korça (Fig. 3, qytet me një bërthamë vernakulare). Megjithatë, karakteristikat e tyre nuk mund të shihen të ndara, pasi në kampionët e marrë në çdo qytet do vërehet larmishmëria e tyre, pasi indin vernakular, komunist dhe informal e gjejmë kudo.

Figure 1 . Qyteti i Tiranës, metropol, me rrjet të dendur dhe mbivendosje formash urbane. Burimi: Autori

Figure 2. Qyteti i Fierit, i zhvilluar kryesisht gjatë periudhës socialiste. Burimi: Autori

Figure 3. Qyteti i Korçës, me një bërthamë vernakulare. Burimi: Autori

Duke qenë se në thelb të analizës do të jenë rrjetet, observimi i tyre lidhet edhe me një aspekt tjetër, llojshmërinë e rrjeteve urbane, duke përfshirë të gjithë strukturën ku mund të kryhet lëvizje, si rrjetet e këmbësorëve, të mjeteve apo biçikletave. Ky studim, për shkak të kufizimeve të softit të përdorur, nuk tenton që ti ndajë ato, por i trajton të gjitha si rrjete urbane që gjenerojnë lëvizje. Gjithashtu, përjashtohet zhvillimi i eksperimentit në kohë, për të vlerësuar ndryshimin e strukturës së rrjeteve.

1.2 Adresimi i problematikës

Së pari, siç u përmend edhe më sipër, në kontekstin shqiptar janë shumë të pakta studimet e realizuara për sa i përket formës urbane, dhe aq më pak lidhja e formës urbane me rrjetet përmes teorive të kompleksitetit. Ndërkohë, që në botë ka gjithmonë e më shumë një vëmendje të shtuar në teoritë e rrjeteve, të lidhura jo vetëm me formën urbane, por edhe me transportin, flukset, lidhshmërinë apo dendësinë. Zhvillimi i softeve të thjeshta, në ndihmë të projektuesve, planifikuesve apo kërkuesve, është një përpjekje për të nxitur zhvillimin e kësaj fushe.

Së dyti, ka një tendencë për braktisje të disa prej qendrave të qyteteve. Njohja e karakteristikave lokale në aspektin e rrjeteve, na përball me nevojën për ti kthyer ato në organizma të jetueshëm.

Së treti, në planet e përgjithshme vendore, si instrumente planifikimi në zbatim, mungojnë analizat dhe perspektivat e zhvillimit të rrugëve në nivel qyteti. Ato kryesisht janë në nivel makro, të lidhura kryesisht me rritjen e dendësisë. Kjo e fundit nuk mjafton për një analizë më komplekse, patjetër që duhet të plotësohet edhe me lidhshmërinë apo aspekte të tjera strukturore të tyre.

1.3 Qëllimi i kërkimit

Fokusi kryesor i këtij kërkimi është hulumtimi i formës urbane për tu përfshirë në ciklin e ripërtëritjes adaptive në qytetet shqiptare, duke u bazuar në analizën e rrjeteve përmes teorisë

së grafeve, të ndërlidhura këto me teoritë e kompleksitetit. Analiza do të shtrihet në tre nivele dhe synon objektivat si më poshtë.

- Analizimi nëse qytetet shqiptare, shfaqin karakteristika të përbashkëta të krahasueshme në leximin e formës urbane, parë në optikën e teorisë së rrjeteve. Përmes softit të përdorur, të bazuar në teorinë e grafeve, të analizojmë të dhënat metrike dhe strukturore, duke i grupuar sipas dendësisë, lidhshmërisë dhe qendërsisë.

- Grupimi i kampionëve me gjeneza të njëjta dhe unifikimi i treguesve me qëllim zbulimin e karakteristikave të tyre të përbashkëta.

- Identifikimi i parametrave ku rrjeti urban shfaq qëndrueshmëri dhe karakter ripërtëritës, me qëllim përmirësimin në nivel aplikativ në fushën e planifikimit.

Ky studim është në vazhdën e studimeve të tjera sipas teorive alternative të morfologjisë urbane, dhe është i pari studim qe analizon rrjetet, nën optikën e teorive të kompleksitetin si fenomen në qytetet shqiptare. Ky koncept i hap rrugën zhvillimeve të mëvonshme mbi tendencat e zhvillimit dhe futjen e këtyre koncepteve në instrumentet planifikuese.

1.4 Hipotezat dhe pyetjet e kërkimit

Hipoteza e ngritur e këtij disertacioni është se leximi strukturës së rrjeteve rrugore, në aspektin e densitetit, lidhshmërisë dhe qendërsisë, krijon një optikë në leximin e formës urbane në qytetet shqiptare. Nën-hipoteza e saj, është se lidhshmëria dhe qendërsia demonstrojnë karakterin ripërtëritës në kohë të tyre. Së dyti, çdo qytet shfaq formën e tij, si atribut të gjenezës së krijimit të tij.

Ky studim, nëpërmjet metodave sasiore të ofruara nga *OSMnx*, mat dhe bën të krahasueshme parametrat në dy nivele, në nivel qyteti dhe në nivel kampionësh urban. Këto parametra nuk kanë vlera unfikuese, por kanë kuptim në nivel lokal. Studimi përpiqet tu përgjigjet pyetjeve të kërkimit si më poshtë:

- A reflektojnë karakteristika të krahasueshme indet urbane shqiptare në qytet të ndryshme?
- A duhet të jenë ndërhyrjet në rrjetet urbane të paramenduara dhe pjesë e planifikimit gjithëpërfshirës apo janë më mirë si produkt i ndërhyrjeve për zhvillim adaptiv nga individi?
- A garanton qëndrueshmëri struktura e rrjeteve në indet vernakulare?

1.5 Qasja eksperimentale

Në këtë kërkim përqendrohemi tek studimi i formës urbane nën optikën e teorive të rrjeteve, (Southworth & Ben-Joseph, 1997), (Fetch, 2012), (Strano, et al., 2013). Në kohë dhe me mungesën e digjitalizimit, analizimi i rrjeteve ka pasur kufizime për sa i përket bazës së të dhënave, madhësisë së kampionëve, thjeshtim i konsiderueshëm i strukturës së rrjeteve dhe

mungese e mjeteve kërkimore të qëndrueshme për tu përdorur, (Boeing G. , 2017a). Për zhvillimin e metodologjisë dhe adresimin e sfidave në këtë disertacion është përdorur soft-i *OSMnx*[3], i cili është një bashkim i *OpenStreetMap*[4] dhe *NetworkX*[5], dhe vjen si një paketë e koduar në *Python*[6], e hapur për përdoruesit. Gjithashtu, përveç *NetworkX*, moduleve të *Python* i shtohen edhe librari të tjera si *Matplotlib, Numpy, Pandas e Geopandas*[7]. Ky soft është krijuar nga Joeffrey Boeing (2017a), drejtues i Laboratorit Urban Data, Universiteti Berkley, Kaliforni, Shtetet e Bashkuara të Amerikës. Qëllimi i këtij soft-i është mbledhja e të dhënave në mënyrë të thjeshtë e të qëndrueshme, bazuar në perspektivat e teorisë së grafeve, transportit dhe projektimit urban.

OSMnx kontribuon në pesë drejtime kryesore: 1) shkarkimi automatik i të dhënave sipas kufijve administrativ të qyteteve, apo kërkesa të tjera të përdoruesit, si dhe gjurmët e ndërtesave brenda tyre; 2) shkarkimi i paracaktuar sipas nevojave dhe ndërtimi i automatizuar i të dhënave të rrjetit rrugor nga *OpenStreetMap*; 3) korrigjimi algoritmik i strukturës së rrjeteve; 4) aftësia për të ruajtur skedarët e rrjeteve si *shapefile*[8], *GraphML*[9] ose *SVG*[10]; dhe 5) aftësia për të analizuar rrjetet, duke përfshirë llogaritjen, projektimin dhe vizualizimin e rrjeteve, dhe llogaritjen e përmasave metrike dhe strukturore. Baza e krijuar nëpërmjet këtij softi, përfshin matje të thjeshta që zakonisht ndodhin në studimet urbanistike dhe të transportit, si dhe matje më të avancuara të strukturës së rrjeteve, të bazuara në teorinë e grafeve (Boeing G. , 2017).

Pas përzgjedhjes së kampionëve, nëpërmjet softit do të realizohen matje metrike dhe strukturale të formës urbane nën optikën e shkencave të rrjeteve. Vlen të përmenden, atributet metrike dhe strukturale janë ndërlidhura me njëra tjetrën (Massuci, D, A, & M, 2009). Analiza do të zhvillohet në këto drejtime:

- Densiteti dhe në këtë studim i referohet mesatares së gjatësive të rrugëve, dendësisë së nyjës, kryqëzimit, skajit apo rrugës. Këto parametra të densitetit janë tregues se sa i dendur dhe i imët është rrjeti.
- Lidhshmëria, duke përfshirë elementë si qarkullimi mesatar, shkalla mesatare e nyjeve, mesatarja e rrugëve për nyje, densiteti i nyjeve për km^2, densiteti i kryqëzimeve për km^2 dhe raporti i laqeve të mbyllura. Këto parametra tregojnë shkallën e lidhshmërisë së rrjetit.
- Qendërsia e nyjeve, tregues për të identifikuar ndërhyrjet nga lart-poshtë kundrejt ndërhyrjeve poshtë-lart të vetë-organizimit dhe evoluimit të strukturës urbane

[3] Soft për analizimin e rrjeteve rrugore dhe gjenerimin e të dhënave.

[4] Është një bazë hartografike e hapur për të gjithë, e modifikueshme, e cila të lejon akses falas në imazhet e saj dhe të gjitha të dhënat e tjera që disponon. Ajo po ndërtohet e përditësohet sipas Informacionit Gjeografik Vullnetar.

[5] Është një paketë për gjuhën e programimit *Python* që përdoret për të krijuar, manipuluar dhe studiuar. strukturën, dinamikën dhe funksionet e rrjeteve grafike komplekse.

[6] Gjuhë programimi.

[7] Programe të koduara në Python, për zhvillimin e analizave gjeo-hapësinore.

[8] Një format i thjeshtë jotopologjik, me anë të të cilit mund të ruhet vendndodhja gjeometrike dhe atributet .gjeografike. Këto të fundit mund të përfaqësohen me pika, vija ose poligone (zona).

[9] është një format skedari i bazuar në XLM, për zhvillimin e grafeve.

[10] Është një skedar vektorial që ruan shkallën e vizualizimit.

- Orientimi i rrugëve, Konfigurimi i rrugëve dhe orientimi i tyre na ndihmon të përcaktojmë rregullat hapësinore dhe logjikën e zhvillimit të qyteteve

Pas vlerësimit të të dhënave të gjeneruara, bëhet analiza krahasuese në nivel qytetesh dhe kampionësh urbanë, për t'i dhënë përgjigje pyetjeve të kërkimit.

1.6 Përmbledhja e kapitujve

Në mënyre të përmbledhur, këtu pasqyrohet një përshkrim i kapitujve që do të ndërtojnë këtë tezë studimore.

Kapitulli i parë: Në kapitullin e parë bëhet prezantimi i objektit të kërkimit, i organizuar në gjashtë seksione si në vijim: sfondi i përgjithshëm dhe konteksti lokal, adresimi i problematikës, qëllimi i kërkimit, hipotezat dhe pyetjet e kërkimit, qasja eksperimentale dhe përmbledhje e kapitujve. Në këtë kapitull bëhet një prezantim i shkurtër me konceptin e kompleksitetit që karakterizon qytetin dhe linja teorike që lexon formën urbane nëpërmjet teorisë së rrjeteve. Më pas, përshkruhen studime të tilla të mëparshme në Shqipëri, dhe nevoja për plotësimin e sfondit kërkimor në këtë fushë. Ky studim ngre hipotezën se leximi strukturës së rrjeteve (densitet, lidhshmëri dhe qendërsi), krijon një optikë në leximin e formës urbane në qendrat e qyteteve shqiptare. Nën-hipoteza e saj, është se lidhshmëria dhe qendërsia demonstrojnë karakterin ripërtëritës në kohë të tyre. Hipoteza e dytë është se çdo qytet shfaq formën e tij, si atribut të gjenezës së krijimit të tij.

Kapitulli i dytë: Në këtë kapitull analizohen konsideratat teorike lidhur me konceptet e formës urbane dhe konceptit të ripërtëritjes adaptive. Në përpjekje për të lexuar drejtë fenomenin e ripërtëritjes urbane, vendos në komunikim disa fusha teorike që konsiderohen të përshtatshme për hapjen dhe adresimin e këtij kufiri të kërkimit. Pas prezantimit teorik të formës urbane, elementëve përbërës dhe shkollave të studimit të saj, nis zgjerimi i konceptit të kompleksitetit. Më pas janë zhvilluar konceptet teorike të matjes së kompleksitetit të formës urbane. Në vazhdim është trajtuar komponenti që analizohet më gjerësisht, rrjetet. Teoria e grafeve, shërben si bazë për studimin dhe zhvillimin e eksperimentit.

Kapitulli i tretë i dedikohet metodologjisë dhe krijimit të bazës së të dhënave që do të përdoren në këtë studim. Si fillim ngrihet diskutimi mbi përzgjedhjen e kampionëve urban, duke i ndarë në dy shkallë të observimit, njëherë në nivel qyteti dhe më pas në nivel kampionësh. Gjithashtu, qartësohet përdorimi i softit dhe dizajni i eksperimentit, duke marrë në konsideratë kufizimet e tij dhe rezultatet e pritshme. Së dyti analizohet modeli teorik i llogaritjes së matjeve metrike dhe strukturore të rrjeteve, duke i grupuar në tre komponentë kryesorë: densitet, lidhshmëri dhe qendërsi. Në këtë kapitull sqarohet teorikisht pse kombinimi i këtyre parametrave jep një këndvështrim mbi interpretimet e formës urbane. Në vazhdim bëhet një prezantim në nivel përshkrimor i rrjetit dhe zhvillimit urbanistik të qyteteve të zgjedhur.

Kapitulli i katërt do të trajtojë produktet e kampionëve të marrë në studim, bazuar në metodologjinë dhe eksperimentin e realizuar. Ky kapitull ndahet në dy pjesë. Në pjesën e parë

zhvillohet eksperimenti në nivel qytetesh dhe bëhet analiza krahasuese midis tyre. Në pjesën e dytë përzgjidhen kampionë nga çdo qytet dhe gjenerohen treguesit përkatës. Më pas, evidentohen në bazë të atributeve të ngjashme dhe origjinës që kanë, kampionët dhe grupohen në tre grupe: vernakular, socialist dhe informal. Për secilën prej analizave, janë hartuar diagrama dhe grafikë krahasues, me qëllim lehtësimin e analizës.

Kapitulli i pestë do të kuptojë rezultatet në këtë proces dhe nxjerrjen e përfundimeve unifikuese, për të lexuar më saktë formën urbane të qyteteve shqiptare. Pas analizës krahasimore në nivel qytetesh dhe më pas në nivel kampionësh për tu përgjigjur saktë pyetje të kërkimit, lind nevoja e krahasimit të kampionëve urbanë me karakteristika të përbashkëta. Krahasimi i tyre mund të gjenerojë vlera të unifikuara për leximin e strukturës së rrjeteve.

Kapitulli i gjashtë paraqet çështjet për përmirësimet në nivel teorik dhe aplikativ dhe pikat e forta të këtij studimi e rekomandime të mëtejshme.

Në përfundim, janë vendosur respektivisht lista e figurave, tabelave, bibliografia dhe anekset e gjeneruara nga softi.

KONSIDERATAT TEORIKE 2

2. KONSIDERATAT TEORIKE

2.1 Forma urbane

2.1.1 Parantezë, këndvështrimi historik i formës urbane

Termi "*formë urbane*" përdoret për të përshkruar karakteristikat fizike të qytetit, duke marrë në konsideratë formën, madhësinë dhe konfiguracionin e tij dhe të elementëve që e përbëjnë atë. Forma urbane është e lidhur ngushtë me shkallën urbane dhe përshkruhet si "*atribut morfologjik i një zone urbane në të gjithë shkallët e saj*" (Jenks, Williams, & Burton, 2000). Për këtë arsye, karakteristikat e saj, variojnë nga një shkallë shumë e lokalizuar, si fasadat apo çarjet e saj, e deri në një shkallë më të madhe, si lloji i strukturës, lloji i rrugës dhe rregullimi i tyre hapësinor. Megjithatë, duhet të theksohet se forma urbane nuk lidhet thjesht me tiparet fizike, por gjithashtu përfshin aspektet jo fizike, përfshirë këtu madhësinë, formën, shkallën, dendësinë, përdorimet e tokës, llojet e ndërtesave, paraqitjen e bllokut urban dhe shpërndarjen e hapësirave të gjelbra. Pra, të marrë së bashku dhe të ndërlidhur me njëri tjetrin, këta komponentë përbëjnë formën urbane të një qyteti. Gjithashtu, duhet thënë se këta elementë janë identifikuar bazuar në mendimin se ndikojnë në qëndrueshmërinë urbane dhe sjelljen njerëzore. Nga shkollat e mendimit mbi morfologjinë urbane, mund të përmendim katër qasje që ndihmojnë në studimin e formës dhe vlejnë të studiohen: **qasja historiko-gjeografike, qasja proces-tipologjike, sintaksa e hapësirës dhe analiza hapësinore.** Të gjitha këto qasje përfshijnë një sferë të gjerë të teorive, koncepteve dhe metodave të tyre në mënyrën e adresimit të formës fizike të qyteteve (Oliveira V. , 2016), (Oliveira V. , 2019).

Morfologjia urbane është studimi i formës fizike të qyteteve si një habitat njerëzor. Nëpërmjet adaptimit që ndodh, ajo shpjegon se si agjentë të ndryshëm mund të ndryshojnë formën fizike në kohë dhe se si procese të larmishme përfshihen në këtë transformim. Forma urbane i referohet elementeve kryesore fizikë që strukturojnë dhe formojnë qytetin duke përfshirë rrugët, hapësira publike, blloqet, parcelat dhe ndërtesat. Në këtë mënyrë, morfologjia urbane mund të jetë një mjet shumë i përshtatshëm për të lidhur elementët teknikë të qëndrueshmërisë së qytetit me aspekte të ndryshme të jetës së qytetit. Pra, bazuar në këtë lidhje, ne mund të ndërtojmë morfologjinë urbane për të treguar se si mënyra e ndërtimit dhe organizimit të qyteteve tona ka ndikim në jetën tonë urbane dhe në aspektet e saj. Batty dhe Longley (1994) kanë vënë në dukje që "*forma e një objekti është një diagram i forcave*" dhe për këtë arsye, çdo studim i formës së qytetit duhet të përfshijë njohuri mbi proceset që e përcaktojnë këtë formë. Në një këndvështrim më modern, forma urbane duhet të përfshijë jo vetëm pjesën e prekshme, ndërtesat por edhe atë jo materiale, si njerëzit apo rrjetet.

Qëllimi i analizës së formës urbane shkon përtej parashikimit të thjeshtë të atributeve fizike të qytetit dhe fokusi ndryshon në varësi të perspektivës që trajtohet. Qasja tradicionale synon të jap kuptim gjeometrisë urbane, e cila interpretohet si rezultat i proceseve historike, ngjarjeve dhe vendimmarrjeve në kohëra. Analiza e strukturës urbane zbulon faza të ndryshme në të cilat është ndërtuar qyteti, modelet e tij hapësinore marrin më shumë kuptim dhe bëhen edhe më të lexueshme. Një nga komponentët ndihmës në kuptimin e formës urbane është komponenti

shpjegues i saj. Ky komponent vlen për të kuptuar në kohë qytetet dhe për të pasur një shqyrtim të marrëdhënies midis proceseve dhe formës hapësinore të cilat kanë kontribuar në formimin e strukturave urbane bashkëkohore. Këto njohuri mund të përdoren për të ndërtuar një teori të projektimit, duke u bazuar në proceset tradicionale të ndërtimit të qytetit (Moudon, 1997) ku sipas kësaj qasje, siguron koherencën e nevojshme tipologjike të transformimeve urbane. Ky ishte dhe objektivi i shkollës italiane për të rikthyer historinë në arkitekturë dhe në projektimin urban, duke synuar që të theksojë gjenezën e krijimit, të lidhur me formën dhe performancën që ajo shfaq. Në këtë mënyrë identifikohen parametra kryesorë që ndihmojnë në shpjegimin se si qytetet marrin një formë të caktuar dhe se si funksionojnë ato. Ajo që vlen të theksohet është se studimi i lidhjes midis formës dhe funksionit ka vlerë të rëndësishme në formën urbane pasi, nëse kjo marrëdhënie është e qëndrueshme atëherë ajo kontribuon në përmirësimin e performancës urbane.

Studimi i formës urbane është nga fushat më klasike të kërkimit urban dhe në gjysmën e dytë të shekullit të XX-të u bë një disiplinë e veçantë teorike. Në këtë periudhë, u vërejtën kontributet inovative të studimit të formës urbane, të stimuluara nga përmirësimet matematikore dhe digjitale. Nëse e vlerësojmë në kohë, këndvështrimet për formën urbane kanë pasur qasje të ndryshme duke filluar nga qyteti ideal i Vitrubius[11] tek analizat tipologjike të Muratorit, e deri tek qasja jomateriale "qyteti i flukseve" i trajtuar ndër të tjerë nga Manuel Castells (2009). Megjithëse këndvështrimet janë të pafundme, ato të gjithë bien dakord në një pikë, që leximi i formës urbane do të thotë të kuptosh më mirë natyrën e brendshme të qyteteve.

Ndër të tjera mund të thuhet se, shekulli i XX-të quhet pa dyshim epoka e urbanizmit dhe jetës urbane, periudha kur qytetet kanë përjetuar zgjerimet e tyre më të mëdha gjatë historisë. Etnografi Levi-Strauss (1955) e përshkruan qytetin si "*shpikja më komplekse e njerëzimit, në bashkimin e natyrës dhe artefaktit*". Qyteti mund të përshkruhet si një integrim i shumë veprimeve individuale dhe në grup, të udhëhequra nga traditat kulturore dhe të formuara nga forcat sociale dhe ekonomike përgjatë kohës. Studiuesit e morfologjisë urbane, bazohen në elementë të rëndësishëm të analizës morfologjike si kopshtet, rrugët, parqet, monumentet, duke qenë se këta elementë mund të konsiderohen si organizma të cilët ndryshojnë dhe transformohen me kalimin e viteve.

Bazuar në diskutimet e Seminarit Ndërkombëtar të Formës Urbane[12] (2016) është arritur në përfundimin se morfologjia urbane mund të kategorizohet nga tre shkolla kryesore: Britanike, Italiane dhe Franceze. Gjithashtu, janë identifikuar prania e shumë studiuesve të tjerë individual që kanë kontribuar në mënyra të ndryshme në këtë fushë, për shembull, MRG Conzen, (1960), një studiues gjerman i njohur për një nga studimet e tij të hollësishme rreth Alnwick, referuar Williams (2000) revolucioni sasior të pasluftës në gjeografi, i cili kaloi në një hulumtim induktiv dhe empirik si mungesë e fuqisë dhe forcës parashikuese (Conzen M. , 1960). Falë tij sot studiohet një prej qasjeve përbërëse të morfologjisë urbane: qasja historiko-gjeografike, e cila bazohet në fokusin kryesor të shkollës britanike.

[11] Arkitekt dhe inxhiner romak

[12] (http://www.urbanform.org/, 2016)

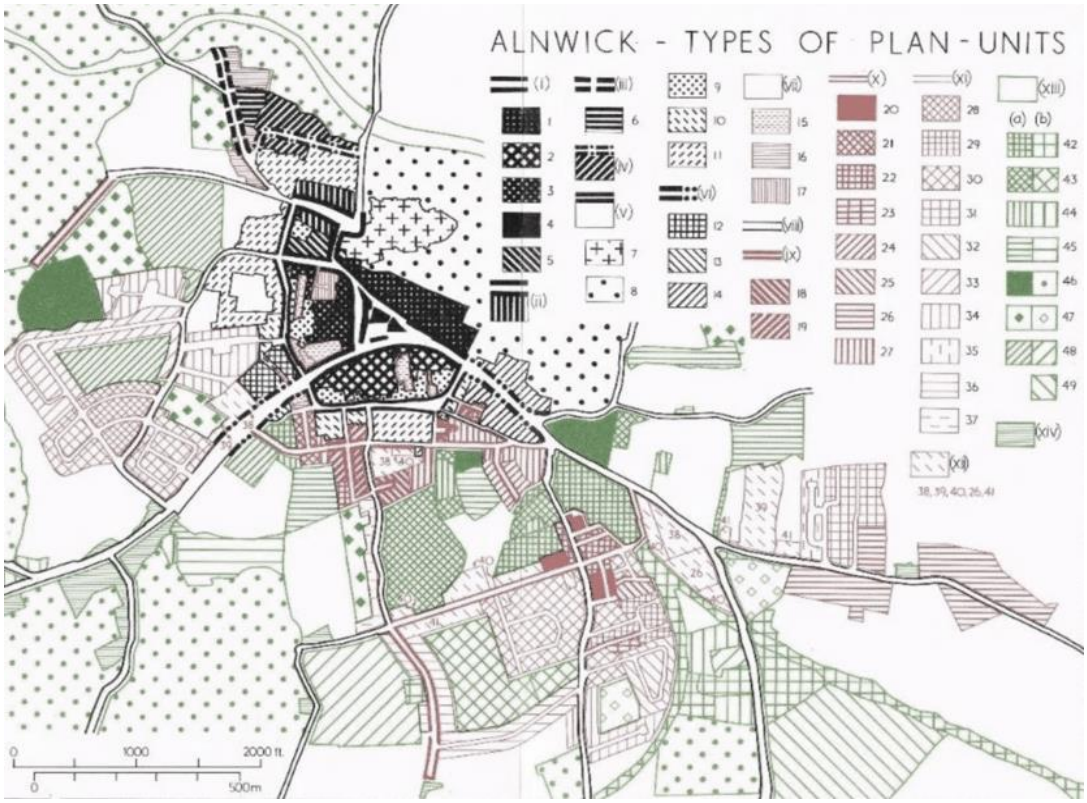


Figure 4 Zonat e periferisë urbane të Alnwick. Burimi: (Conzen M. , 1960)

Nga ana tjetër, Saverio Muratori (1910-1973) arkitekt italian i cili ka përdorur termin e titulluar "*historitë operacionale*" të Venecias dhe Romës, të cilat përbëjnë dhe bazën teorike për studimin e projektimit arkitektonik, (Muratori, 1959). Ai kundërshtoi praktikën e ashtuquajtur "*tabula rasa*" në zonat historike dhe theksoi se analiza morfologjike e qytetit ekzistues duhet të ishte një pjesë e detyrueshme e studiove të projektimit (Mudon, 1994).

Figure 5. Studimi morfologjik mbi Venecian nga Saverio Muratori, themeluesi i shkollës së morfologjisë italiane. Burimi: (Muratori, 1959)

Sidoqoftë, pikat e forta të mësimeve të Conzen dhe Muratori-t tërhoqën pasuesit e tyre, që vunë re rëndësinë e asaj që mjeshtrit e kishin quajtur "*genius loci*[13]" të qytetit, dhe rendësinë e memories kolektive e kulturore. J.W.R Whitehand siguroi trashëgiminë e Conzen duke përpiluar disa nga punimet e tij dhe duke zhvilluar më tej domethënien e ideve të tij (J & Conzen, 1981). Whitehand i drejtoi kufijtë e morfologjisë urbane drejt ekonomisë urbane, duke hulumtuar marrëdhëniet midis qytetit, banorëve të tij dhe dinamikës së industrisë së ndërtimit (Whitehand J. W., 2011). Në vitin 1974, ai formoi Grupin Kërkimor të Morfologjisë urbane në Universitetin e Birmingam-it, i cili merrej me kërkime mbi qytetet mesjetare, si dhe studime mbi zgjerimet dhe transformimet periferike të shekullit XX-të. Një program i qëndrueshëm i konferencave dhe publikimeve gjatë 25 viteve të fundit, e ka bërë këtë grup një qendër jashtëzakonisht të fortë kërkimi, duke vazhduar traditat kryesore në gjeografinë urbane. Gjithashtu, të diplomuarit në doktoraturë në Birmingam, si Peter Larkham, Karl Kropf dhe Keith Lilley, kanë ndihmuar në përhapjen e ndikimit të këtij grupi.

Në mësimet dhe botimet e tij, Caniggia vazhdoi traditën *muratoriane*, të cilën e quajti "*proces-tipologjike*" për shkak të përqendrimit në ndërtimin e llojeve, si thelbi i formës urbane. Kjo gjithashtu konsiderohet si qasja e dytë në studimin e formës urbane, ajo e proces-tipologjisë që përfshin tre koncepte themelore, si: rruga, ndërtesa themelore dhe ndërtesa speciale (Strappa, 2018). Ashtu si Muratori, Caniggia e vuri në praktikë teorinë e tij, duke qënë tëpër aktiv në arkitekturë dhe në ndërtim gjatë gjithë jetës së tij. Kërkimi i tij u shtri në disa qytete në Itali dhe Afrikën e Veriut, të kryera me kolegë dhe studentë që vazhdojnë trashëgiminë Muratoriane. Sot, Giancarlo Cataldi, Gian Luigi Maffei, Maria Grazia Corsini, Paolo Maretto, Giuseppe Strappa dhe të tjerë, vazhdojnë traditën në Firence, Romë, Gjenova dhe në Siena. Pasi Caniggia dhe Muratori kishin krijuar terrenin për dy shkollat më të hershme të morfologjisë urbane, një shkollë e tretë u shfaq në Francë në fund të viteve 1960, kur arkitektët Philippe Penerai dhe Jean Castex, së bashku me sociologun Jean-Charles De Paule, themeluan Shkollën e Arkitekturës në Versajë si pasojë e shpërbërjes së Arteve të Bukura. Ashtu si Shkolla Italiane, shkolla Franceze u ngrit nga një rrymë kundër arkitekturës moderniste dhe refuzimit të saj ndaj historisë. Sidoqoftë, ajo gjithashtu përfitoi në atë kohë nga diskursi i gjallë intelektual mbi jetën urbane, i cili tejkaloi arkitekturën dhe angazhoi kritikë të tillë të fuqishëm si sociologu Henri Lefebvre dhe historianët e arkitekturës Francoise Boudon dhe André Chastel. Teksa merreshin me kërkime mbi evolucionin historik të lagjeve parisiane, Panerai dhe Castex studiuan veprat e Muratorit, të panjohura në Francë në atë kohë, gjë që i siguroi shtysën për të hetuar më tej dimensionet teorike dhe metodologjike të punës së tyre. Me kalimin e viteve, ata vendosën kontakte me studiuesit e tjerë që ndodheshin jo vetëm në Itali, por edhe në Spanjë dhe Amerikën Latine. Produktet e këtyre shkëmbimeve mbeten ende për t'u dokumentuar. Nga ana tjetër, botimet e hershme të Castex dhe Panerai ushtruan ndikim të konsiderueshëm në të gjithë komunitetin e arkitektëve evropian. Studimet e hollësishme pasuese të qytetit të Versajës, bastideve franceze dhe qytetit të Kajros dhe Egjiptit, ndihmuan në përgatitjen e një brezi të dytë morfologësh në Francë. Gjatë viteve 2000, grupet kërkimore janë themeluar në Nant, nga Michael Darin dhe në Marsejë, nga Jean-Lucien Bonillo. Ky bashkim i studiuesve nga zona dhe disiplina të ndryshme gjuhësore është i themeluar në baza të përbashkëta.

[13] t'i japësh një vendi autenticitet

Së pari, ekziston marrëveshja që qyteti mund të *'lexohet'* dhe të analizohet përmes mesatares së formës së tij fizike. Për më tepër, pranohet gjerësisht se në nivelin e saj më elementar, analiza morfologjike bazohet në tre parime (Moudon, 1997):

i. Forma urbane përcaktohet nga tre elementë themelorë fizikë: a) ndërtesat dhe hapësirat e tyre të lidhura, b) parcela ose toka, dhe c) rrugët;

ii. Forma urbane mund të kuptohet në nivele të ndryshme të elementëve përbërës. Zakonisht, katër janë të njohura, që korrespondojnë me ndërtesën / parcelën, rrugën / bllokun, qytetin dhe rajonin;

iii. Forma urbane mund të kuptohet vetëm historikisht pasi elementet prej të cilave përbëhet, i nënshtrohen transformimit dhe zëvendësimit të vazhdueshëm.

Kështu forma, zgjidhja dhe koha përbëjnë tre përbërësit themelorë të kërkimit morfologjik urban.

Parimet e mësipërme janë të pranishme në të gjitha studimet, qoftë nga gjeografë apo arkitektë, por kanë shërbyer edhe nëse analiza përqendrohet në një qytet mesjetar, barok apo bashkëkohor. Njësia më e vogël e qytetit njihet si ndërthurja e dy elementeve të saj: parcela e tokës me ndërtesën ose ndërtesat hapësirat e hapura të saj. Karakteristikat e njësisë më të vogël përcaktojnë formën dhe dendësinë e formës urbane, si dhe përdorimin e saj aktual dhe potencial me kalimin e kohës. Studimet tregojnë se atributet e njësisë më të vogël dhe elementët e saj reflektojnë jo vetëm një periudhë kohore të caktuar, por kushtet socio-ekonomike të pranishme në kohën e zhvillimit dhe ndërtimit të tyre. Me kalimin e kohës, këto njësi bazë të formës urbane ndryshojnë përdorim, fillimisht përdoren nga një klasë e caktuar shoqërore, pastaj me tranformimet që pëson qyteti, përdorimi i tyre i kalon një kategorie tjetër njerëzish. Shkalla e ndryshimit, e cila mund të jetë në funksionin e tyre ose në formën e njësisë, ndryshon nga qyteti në qytet, por gjithashtu përgjithësisht përshtaten në cikle që lidhen me ekonominë dhe kulturën. Ciklet e ndërtimit dhe transformimit janë procese të rëndësishme për të eksploruar mbi planifikimin e qytetit dhe qëllimet e zhvillimit të pasurive të patundshme, megjithatë rrallë janë studiuar në qytetet bashkëkohore. Studimet përqendrohen gjithashtu në atë që Conzen e quan "*njësia e planit* 'dhe atë që italianët e quajnë *"pëlhurë urbane"*. Njësitë e planit ose pëlhura urbane janë grupe ndërtesash, hapësira të hapura, blloqe dhe rrugë, të cilat formojnë një tërësi kohezive, ose sepse të gjitha ishin ndërtuar në të njëjtën kohë apo brenda të njëjtave kufizime, ose sepse i janë nënshtruar një procesi të përbashkët transformimi.

Secila nga shkollat ka pasur synime të ndryshme në përpjekjet e saj për ndërtimin e teorisë, të cilat mund t'i grupojmë si më poshtë:

i. Propozimet kryesore të 'Shkollës Birmingham', e cila është më e plotë se shkollat e tjera të morfologjisë urbane, është përqendruar në studimin e formës urbane për qëllime përshkruese dhe shpjeguese, për të zhvilluar një teori të ndërtimit të qytetit. Studime të tilla kanë të bëjnë me mënyrën se si janë ndërtuar qytetet dhe pse. Ky është qëllimi kryesor i gjeografëve dhe "Shkollës së Birminghamit" në veçanti. Gjithashtu, shkolla britanike ka një avantazh kur bëhet fjalë për vëmendjen që i kushton ngjarjeve historike.

ii. Për Shkollën Italiane fokusi kryesor është studimi i formës urbane për një qëllim të caktuar përshkrues, duke synuar të zhvillojë një teori të modelit të qytetit, e cila çon në mënyrën se si duhet të ndërtohen qytetet. Ky qëllim është i orientuar në një drejtim të veçantë, përkatësisht për të zhvilluar një teori të modelit të ndërtimit mbështetur në një traditë historike të të ndërtuarit të qytetit. Por, duket se shkolla italiane e mendimit nuk është mjaft e vlefshme për vetë faktin se nuk merr parasysh të kaluarën dhe dimensionet e ndryshme të krijimit të formave të pëlhurave urbane.

iii. Nga ana tjetër, 'Shkolla Franceze' me fokus kryesisht në shkollën e Arkitekturës së Versajës, vendos në qendër të vëmendjes në studimin e saj formën urbane duke vlerësuar ndikimin e teorive të projektimit në ndërtimin e qytetit. Përveç faktit që ky mbetet një ushtrim i vështirë mendor për shumë planifikues, ata kanë arritur të vlerësojnë ndryshimet apo ngjashmëritë midis direktivave të deklaruara rreth asaj që duhet të ndërtohet (teoritë normative) dhe asaj që në të vërtetë është ndërtuar. Shkolla e mendimit francez integron dhe analizon shumë dimensione të arkitekturës, shoqërisë, ekonomisë dhe politikës në të njëjtën kohë (Moudon, 1997).

Këto tri qasje të shkollave, *italiane, angleze dhe franceze*, bashkohen në idenë mbi formën urbane si shprehje fizike të ndryshimeve në forcat që operojnë në qytet. Metodat e tyre analitike janë të bazuara në vëzhgimin e modeleve hapësinore dhe fokusi i studimeve të tyre ishte qyteti historik. Megjithatë evoluimi i jetës, shfaqja e teknologjive të reja dhe përparimet e shpejta në çdo fushë, sollën padyshim zgjerim urban dhe rritje të kompleksitetit të morfologjisë urbane, duke sjellë gjithashtu nevojën e studimit me objektiva dhe mundësi të reja analitike. Kjo solli që dhe vetë fusha urbane të kishte nevojën për përkufizime të reja.

Propozimet për struktura të reja urbane rezultuan në lagje të distancuara dhe të shpërbëra, pasi vëmendja iu ishte dhënë marrëdhënies midis zonave urbane, periferike dhe natyrës. Kufiri midis hapësirës urbane dhe rurale përcakton kështu një tipologji të re klasifikimi të formave të qytetit. Pas disa etapave të hulumtimit, u arritën në analogji të përgjithshme për qytetet, duke i përkufizuar si 'fletë', 'bërthamë', 'galaktike', 'satelitore', etj. (Marshall S. , 2009) apo qyteti 'yll' i Lynch (2002). Megjithatë, elementët e rendit vizual, pavarësisht rëndësisë që kishin pasur përgjatë fillimit të shekullit, tashmë nuk ishin përcaktuesit përfundimtarë të metropolit modern. Forma fizike nuk ishte thelbësore aq sa për të përcaktuar nivelin e përgjithshëm të atraksionit njerëzor dhe kjo solli një krizë në çështjet e planifikimit urban.

Kjo zhvendosje rezultoi gjithashtu në një neutralizim gradual të hapësirës, i cili u zbeh më tej nga shfaqja e teknologjive të reja të informacionit dhe komunikimit. Këto teknologji kthyen nocionin e distancës dhe sfiduan rëndësinë e vendndodhjes duke zbehur edhe me tepër rolin tradicional të qyteteve si vende të ndërveprimit shoqëror dhe shkëmbimit tregtar. Realiteti urban filloi të shpjegohej dhe interpretohej në terma të lexueshëm dhe në shpjegime analitike. Gjithashtu, bazat e të dhënave statistikore zëvendësuan hulumtimet topografike si instrument kryesor analitik. Megjithatë, nga zhvillimi teknologjik është mundësuar që vitet e fundit të realizohet një studim më i detajuar dhe në anën virtuale të eksplorimit të formës urbanë në një gamë më të gjerë çështjesh.

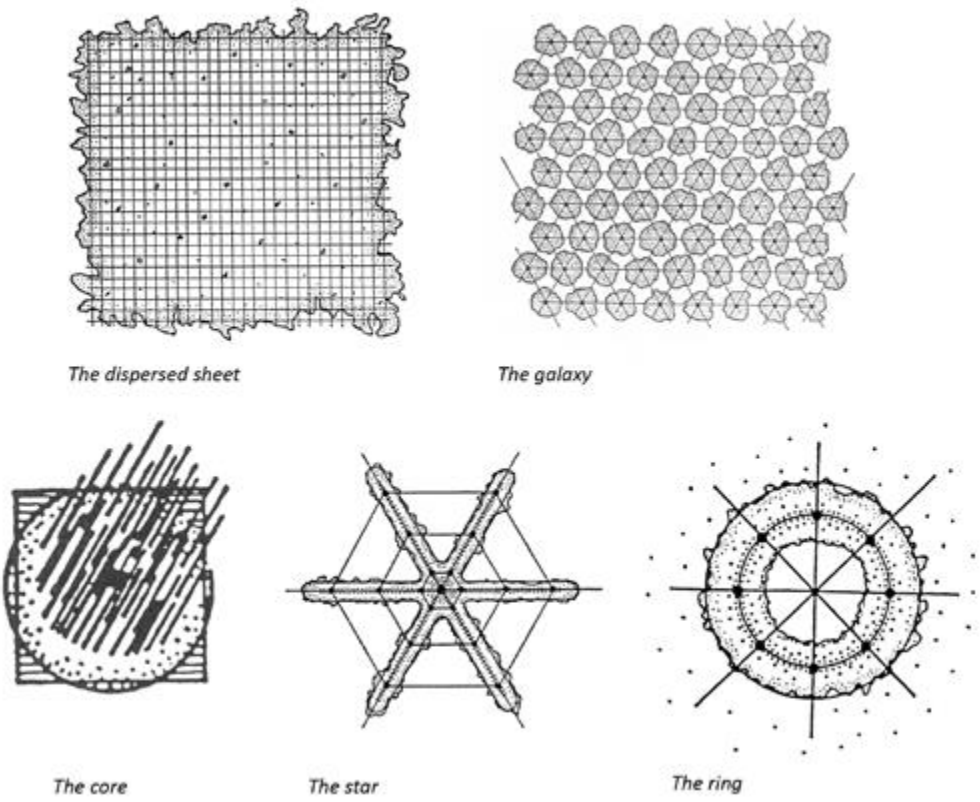


Figure 6 Modulet e formës urbane sipas Lynch. Burimi: (City sense and city design: Writings and projects of Kevin Lynch, 2002)

2.1.2 Tipologjitë dhe Indi Urban

Megjithëse teoritë *muratoriane* nuk krijuan një përmbledhje holistike, metoda e tij shtjelloi identifikimin e llojeve themelore historike të strukturës urbane. Ai e përcaktonte tipin si një prioritet sintetik i karakteristikave të lidhura të objektit (Caniggia & Maffei, 2003). Kjo metodë analitike mundësonte kuptimin e qytetit si një bashkim i përbërësve në shkallë të ndryshme.

Gjurma është njësia bazë strukturale, e cila ka një kombinim të atillë tipologjik i cili përcakton karakterin e indit urban. Indi, nga ana tjetër, është ekuivalent i tipologjisë, por në një shkallë më të gjerë dhe bashkimi i tij përbën organizimin urban në tërësi. Caniggia gjithashtu punoi mbi klasifikimet e ndryshme për gjurmët, indet dhe sistemin urban, duke përfunduar në një logjikë pas konfigurimeve fizike ekzistuese (Caniggia, Maffei, & Galán, 1995).

Tipi bazë, korrespondon me nivelin e gjurmës. Ai përfshin ndërtesën dhe hapësirën e hapur që lidhet me të. Këtu mund të përmendim diferencime të bëra midis ndërtesave të veçanta (kishat, tregjet) dhe tipologjitë e banimit, të cilat përcaktojnë karakterin përbërës në varësi të epërsisë së tyre. Studimet e realizuara në qytetet italiane tregojnë dy lloje *tipash bazë*: (i) *Shtëpitë me tarraca, mbizotëruese në Gjenova, Firence dhe Romë dhe (ii) Shtëpi Patio*[14] *e cila mbizotëron në Milano, Bolonja ose Napoli.*

[14] shtëpi me oborr

Indi urban, përcaktohet nga tipologjia mbizotëruese e ndërtesës dhe shtigjet që mundësojnë hyrjen dhe lidhjet. Caniggia përshkruan katër kategori të rrugëve nga të cilat kanë origjinën lloje të ndryshme të indeve urbane (figura 7): (i) *Rrugët si origjinë. Ato kompozojnë indin origjinal urban dhe lidhin polet; (ii) Rrugët e themelimit. Rrugë të reja që janë vendosur në mbushje të mëtejshme pasi është vendosur indi mëmë; (iii) Rrugët e lidhjes janë ato, paraqitja e të cilave i bindet nevojës për akses në ndërtesat ekzistuese, duke formuar blloqe urbane; dhe (iv) Rrugët e ristrukturimit, ato janë të mbivendosura mbi një ind të mëparshëm, duke krijuar një dendësim të strukturës.*

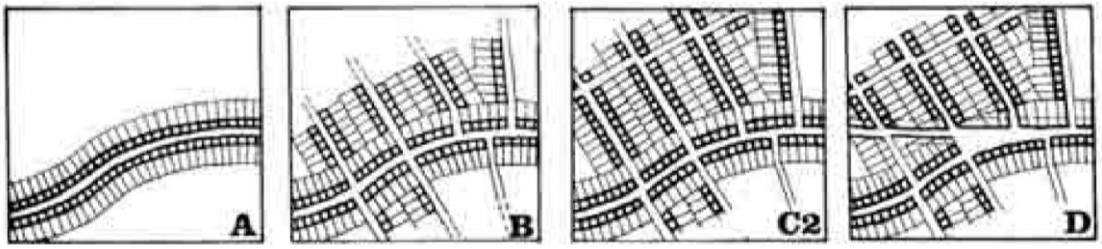


Figure 7 Modele të zhvillimit të pëlhurës urbane a-rrugë si origjinë b-rruga e themelimit c-rruga lidhëse d-rruga e ristrukturimit. Burimi: (Caniggia, Maffei, & Galán, 1995)

Sistemi Urban, përbëhet nga ndërthurja e indeve urbane, por edhe nga një zonë ndikimi përreth saj, e cila mund të përfshijë tokë bujqësore apo vendbanime të tjera që formojnë pjesë të të njëjtit sistem. Hierarkitë që janë themeluar më zgjerimin e qytetit sjellin modele të rritjes modulare, të cilat përmblidhen në tre kategori kryesore (figura 8):

(i) *Rritje qendrore*
(ii) *Rritje në dy drejtime*
(iii) *Lineare*

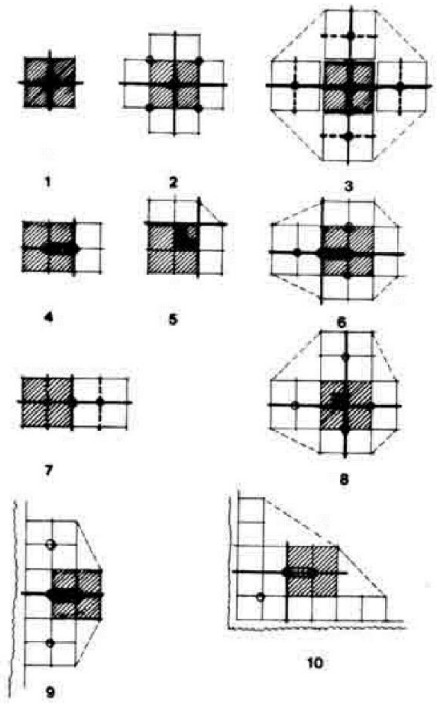


Figure 8 1-moduli bazë; 2-rritje qëndrore; 3&8 Ritje në dy drejtime; 4,5,6,7-rritje lineare. Burimi: (Caniggia, Maffei, & Galán, 1995))

2.1.3 Rajonet Morfologjike

Teksa ndalemi tek shkolla *conzeniane*, mund të themi se foma urbane në këtë rast lexohet si përbërje e shtresëzuar prej tre shtresash: plani i qytetit, masa e ndërtuar tre-dimensionale dhe shfrytëzimi i tokës e ndërtesave (Whitehand J. , 2007). Këto tre shtresa, me qëllim përcaktimin e zonave të tipit homogjen arrinin të përcaktonin atë që Conzen i quante si rajone morfologjike apo njësitë e peizazhit urban të qytetit. Megjithatë, shkolla *conzeniane* mori disa lidhje midis karakteristikave fizike dhe proceseve origjinale që ishin tepër të dobishme në krijimin e rajoneve morfologjike, të ilustruara në qytetin e vjetër të Ludlow në tabelën e mëposhtme. Por, i influencuar nga prejardhja dhe studimet e tij, Conzen evoluoi në konceptin e "*Fringe Belt*[15]" si një entitet dallues, ku nëpërmjet analizës lexohej jo vetëm struktura e qyteteve, por dhe mënyra se si ato duhet të planifikohen (Whitehand J. , 2007).

Table 1 Komplekset e formës dhe karakteristikat hapësinore për të përcaktuar rajonet morfologjike. Burimi: (Conzen M. G., 2004)

1	2		3	4
Sistematik (I rregullt) nga kompleksi	Shkalla e formës së zgjatjes	Periudhat morfologjike	Përbërësit morfologjikë të shtresëzimit historik	Kontributi në hierarkinë e rajoneve të peizazheve të qytetit
Plani I qytetit	Maksimumi	Mesjetar I lartë 1090-1270	Përshkrime të përgjithshme të sistemit rrugor, modelit të parcelës dhe pozicionimit të ndërtesës	Kategori e lartë (njësitë e planit kryesor gjenetik) kategori e ndërmjetme (lagjet; rrugët dhe njësitë në përqindje, periferi të mesjetës së lart.
		Mesjetar I vonë 1270-1500 dhe post-mesjetar I hershëm	Ndërhyrje të mëdha ishullore dhe anësore në tregun e rrugëve ndryshime të kudogjendshme në linjat e rrugëve nga shkelje të vogla anësore, ndryshime të vogla të kudogjendshme në strukturën e parcelës	Kategori e ndërmjetme (transformimi I Dinhamit lindor, lagja Bell Lane) kategori e ulët
Pëlhurë ndërtimi	Të konsiderueshme duke ndryshuar me periudha	Mesjetar I lartë dhe I vonë 1090-1500	Pak ndërtesa publike por të shquara dhe struktura mbrojtëse. Shumë pak shtëpi sipas indekseve të jashtme, por struktura mbetet brenda dhe në pjesën e pasme të shumë shtëpive post-mesjetare	
		Modern I hershëm 1500-1840	Shumica e shtëpive në përzierjen e periudhës së lokalizuar	Kategori e ndërmjetme, por kryesisht kategoria më e ulët
		Viktorian dhe Eduardian 1840-1918	Banesa në vendndodhje periferike ose në rrugë të vogla. Disa ndërtesa tregtare në qendrën e biznesit	Kategoria më e ulët
		Ndër-dhe pas-luftës. Pas 1918	Shumë pak ndërtesa brenda Qytetit të Vjetër	
Shfrytëzimi I tokës urbane		Para-1840	Shumica e zonave të përdorimit të tokës (qendra biznesi, zona rezidenciale, zona institucionale	Kategori mesatare (qendër biznesi, zonë rezidenciale tradicionale, zona rekreative, rrënojat e kalasë)
	Minimal	Së fundmi (shekulli I XX-të)		

[15] Fashë ndarëse

Më tej, shkolla Franceze e morfologjisë, siç është përmendur edhe më lart, kombinoi qasjen italiane me konceptet e Lefevre mbi hapësirën dhe cilësinë e formës, (Lance, 2004). Ajo që parashtron metoda e tyre, është një thyerje midis strukturës së ndërtuar dhe strukturës sociale. Në fazën e parë analitike, metoda fiton një objektivitet "virtual" i cili zbulon logjikën e sistemeve që përbëjnë mjedisin e ndërtuar. Në këtë mënyrë, mund të arrihet mundësia e paraqitjes së aspekteve historike dhe shoqërore në përshkrimet morfologjike të cilat ndahen në retroaktive dhe aktuale (Lance, 2004). Pamjet retroaktive të qytetit klasifikojnë aspektet formale të rritjes urbane në dy kategori:

(i) Rritje e vazhdueshme, që nënkupton dendësi maksimale, kufij të papërcaktuar, pa pengesa fizike dhe qyteti përthith fshatin dhe vendbanimet e tjera nga zgjatimet periodike të strukturës urbane. Kjo rritje e vazhdueshme mund të jetë lineare apo polare, në varësi të shpërhapjes së proçesit të urbanizimit në një apo disa akse.

(ii) Rritje e pavazhduar, i referohet një përshkrimi global të urbanizimit mbi një territor, i cili përfshin qytetin e vjetër, shtrirjen urbane dhe ngastra të ndërmjetme natyrore. Koncepti ka të bëjë më teoritë e Howard (2003) si dhe me qytetet satelitore, si p.sh. në qytete si Londra apo Bath.

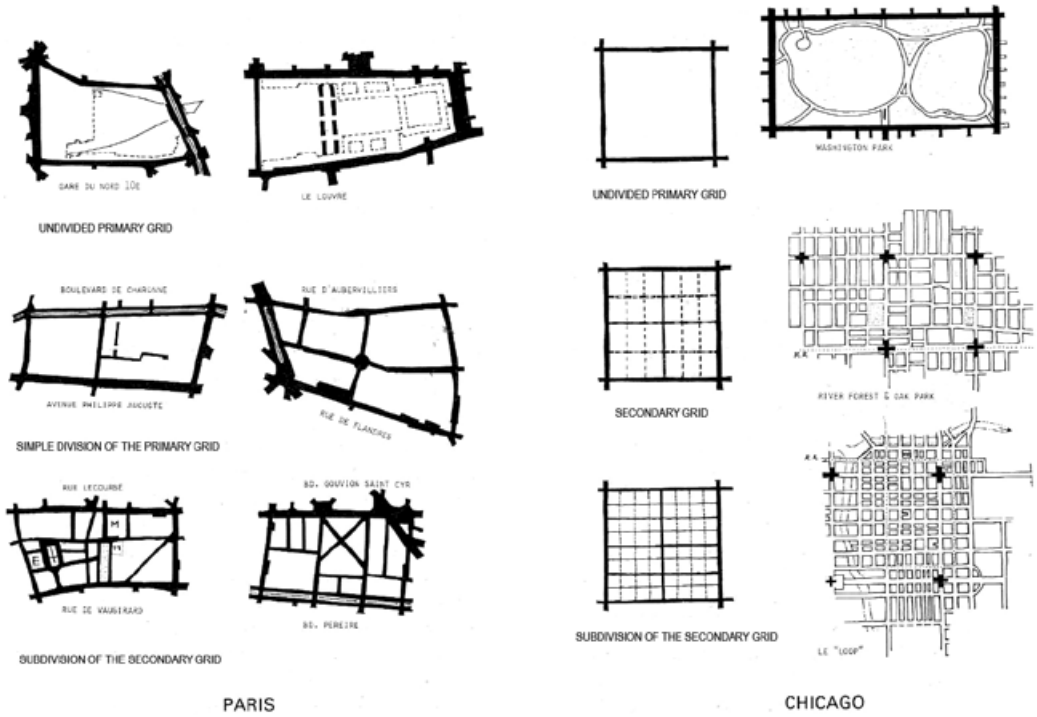


Figure 9 Komunikimi në nivel hierarkik i rrjeteve në Paris dhe Chicago. Burimi: (Panerai, Samuels, Castex, & Depaule, 2004)

Analiza e ndërtimeve në një qytet përbën një nga bazat e njohurisë mbi të, por ajo duhet të pasurohet për të vendosur komunikimin midis mjedisit të ndërtuar dhe atij social. Për shkak të kompleksitetit të lartë të fenomeneve urbane, analiza duhet të thjeshtohet në një sistem të prekshëm dhe mund ta bazojmë në tre elementë kryesorë:

(i) Rrjetet – të cilat formojnë rrjetin bazë të strukturës urbane. Ato mund ti ndajmë në rrjet primar dhe rrjet sekondar, ku ky i fundit lidhet edhe me madhësinë e bllokut të banimit (figura 9).

(ii) Ndërtesat publike – të cilat kanë një rol të veçantë në strukturën urbane, me dy qasje në shkallët e observimit. Në nivel lokal, kanë një funksion të caktuar dhe një hapësirë ndryshe nga ndërtimet e tjera në qytet dhe në shkallë qyteti janë pika kyçe të tij.

(iii) Blloku urban– edhe pse është një element i njohur, nuk është domosdoshmërisht pjesë e strukturës morfologjike, pasi jo gjithmonë janë homogjenë në një qytet. Panerai (2004) e ka cilësuar ndryshimin e shkallës së tij, si një transformim të paprecedentë të shekullit të XIX-të (figura 10). Analizat kritike në këtë pikë, kanë theksuar se prishja e madhësisë së bllokut ka pasoja në përditshmërinë e të përjetuarit të qytetit.

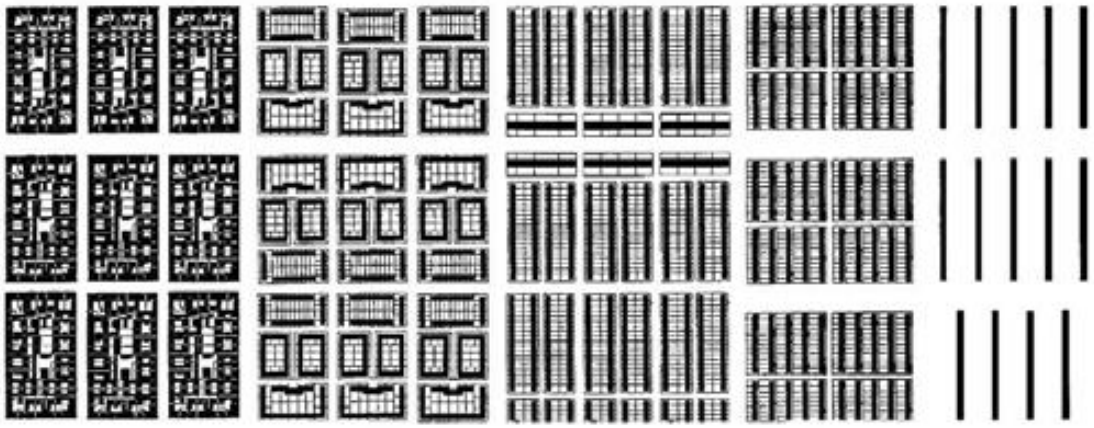


Figure 10 'Thyerja e bllokut urban'. Burimi: (Panerai, Samuels, Castex, & Depaule, 2004)

2.2 Kompleksiteti i zhvillimit urban

Qytetet janë sisteme që bashkojnë një sasi të madhe njerëzish të cilët veprojnë sipas nevojave, dëshirave, aftësive të ndryshme dhe kapaciteteve. Fakti që qytetet konsiderohen si sisteme, vjen si pasojë e përbërjes së tyre: nga grupe sendesh të lidhura apo nga pjesë që formojnë një tërësi komplekse. Vështirësia e kuptimit të sistemeve, pavarësisht se sa të ndërlikuar janë ato, vjen dhe nga ndërveprimet që ekzistojnë midis nën-sistemeve të ndryshme dhe se sa shpejt apo papritur ndryshojnë ato. Në qytete, njerëzit dhe sistemet që ata krijojnë, bashkëveprojnë dhe ndryshojnë me kalimin e kohës, duke sjellë përpjekje në përshtatjen e tyre ndaj ndryshimeve të vogla apo të mëdha që ndodhin në kontekst; pra, kjo i bën qytetet në vetvete komplekse.

Për të kuptuar më mirë sistemet komplekse, disa shembuj të thjeshtë janë: gjuhët, kolonitë e insekteve, truri i njeriut, sistemet ekonomike, sistemi imunitar, moti apo tufat e zogjve. Ndonëse një milingonë e vetme nuk mund të konsiderohet veçmas inteligjente, një koloni e përbërë nga qindra e mijëra milingona mund të punojë së bashku duke u sjellë kështu si një e vetme, si një sistem kompleks inteligjent. Edhe në rastin e përgjithshëm të qyteteve, pa konsideruar rastet e rralla, ekziston një formë e organizmit qendror që planifikon shpërndarjen dhe furnizimin me ushqim ne secilën zonë të tij.

Mitchell (2009, p. 13) e përcakton kështu një sistem kompleks: '*Një sistem kompleks është një sistem në të cilin ka një mori të madhe përbërësish pa një kontroll qendror dhe rregullat e thjeshta të funksionimit krijojnë sjellje kolektive komplekse, përpunim të sofistikuar të informacionit si dhe adaptim përmes mësimit të evolucionit*'.

Ndërsa, Batty (2009, p. 4) i përshkruan ato duke treguar sjellje "*befasuese dhe të paparashikuara ose emergjente siç tregohet në modelet e përgjithshme që lindin, nga funksionimi i proceseve të sistemit në nivelin mikro*"

Në përmbledhje, sistemet komplekse:

(i) Sisteme të mëdha pa kontroll qendror;
(ii) Të përbëra nga përbërës ose agjentë që veprojnë dhe bashkëveprojnë duke përdorur rregulla të thjeshta;
(iii) Veprime që rezultojnë në modele të paparashikuara, pra komplekse në sistemin në tërësi.

Kompleksiteti është një nga aspektet më thelbësore të qytetit të qëndrueshëm dhe projektuesit shpesh diskutojnë mbi formën fizike urbane në terma të tij. Holland (1995, p. 4) parashtroi se të gjithë sistemet komplekse kanë një veti qendrore: aftësinë e tyre për të ruajtur koherencën përgjatë ndryshimeve, pa ndonjë kontroll qendror. Vetë termi kompleksitet i referohet sjelljes së pasur, dinamike të sistemit që lind nga ndërveprimet individuale midis shumë nënpërbërësve heterogjenë (Cilliers, 1998). Objektivi kryesor i kompleksitetit është të përcaktojë disa përbërës të përgjithshëm të sistemeve komplekse të përfshira.

Gjithashtu, morfologët urban kanë studiuar kompleksitetin e formës urbane, koncepte të cilat janë ndjekur edhe nga studiues si Jane Jacobs (1961) dhe Christopher Alexander (1965), mbi paradigmat e ndryshme të projektimit urban që sot adresojnë drejtpërdrejt dhe indirekt për vlerën e kompleksitetit në mjedisin e ndërtuar. Sidoqoftë, pavarësisht faktit se janë bërë shumë vite të kërkimit në këtë fushë, ne ende nuk mund të diskutojmë në lidhje me një teori të vetme të kompleksitetit, por përkundrazi një grup shumë të gjerë konceptesh dhe mjetesh që mund të zbatohen për studimin e kompleksitetit ose sistemeve komplekse (Haken, 2012). Megjithëse përfshin një zhvendosje thelbësore larg besimit se siguria parashikuese është e mundur me sisteme komplekse, ai mund të shërbejë si një optikë e re e dobishme për shpjegimin e fenomeneve urbane, studimin e formës së qytetit dhe shqyrtimin e ndërhyrjeve të planifikimit. Për më tepër, kompleksiteti ofron një kornizë gjithëpërfshirëse për vlerësimin e sjelljes së sistemit që mund të ndërtojë lidhje më të forta midis disiplinave urbane sasiore dhe cilësore (Portugali & Stolk, 2016).

Në shekullin e XX-të, nga Corbusier dhe bashkëkohësit e tij, pati një qëndrim tjetër lidhur me kompleksitetin fizik të qyteteve tradicionale. Ata krijuan konceptet e para për zonimin funksional me një përdorim të vetëm dhe varësinë nga automjetet nëpër qytete (Hall, 1996). Më pas, Scott (2020) kritikoi modelin modern urban, duke i quajtur qytetet utopike nga lart-poshtë, të thjeshta dhe racionale, dhe vlerësoi urbanizimin nga poshtë-lart, duke i konsideruar të ndërtuara organikisht, të çrregullta, të varura nga njohuritë lokale (Jacobs J. , 1961). Projektuesit modern zhvilluan rendin vizual gjeometrik për një funksionim të mirë, me rregull të qëndrueshëm shoqëror në mjedisin e ndërtuar (Sussman & Hollander, 2015). Për më tepër, planifikuesit urbanë gjithmonë kanë diskutuar në terma të "kompleksitetit" kur bëhet fjalë për

formën fizike urbane dhe zhvillimin e saj. Por, në të gjitha studimet dhe qasjet, ekziston një pikëpamje e përbashkët për mënyrën se si ndikon kompleksiteti në habitatet njerëzore në shkallën e lagjes dhe jetueshmërinë. Për me tepër, ne rrethohemi gjithkund nga sisteme komplekse.

Disa forca të brendshme ndikojnë në strukturën ose rritjen e sistemit dhe përbërësit e tij rrisin koherencën e brendshme të strukturës së tij për shkak të atij vet, ky fenomen quhet vetë-organizim (Portugali J. , 2000) (Salingaros N. , 2005).

Ndërkohë, adaptimi (nga latinishtja "përshtatja") është një pjesë vendimtare e sistemeve komplekse. Për ta zgjeruar edhe me shumë, kapaciteti adaptiv është ajo pikë ku qyteti duhet të përshtatet për të qëndruar koherent. Sistemet komplekse ku adaptimi është një fenomen shumë i rëndësishëm, zakonisht referohen si sistemet komplekse adaptive (Holland 1995). Qytetet janë një shembull thelbësor i sistemeve komplekse adaptive. Për rrjedhojë, studimi i qyteteve si sisteme komplekse mund të jetë shumë sfidues për faktin se një qytet më kompleks edhe problemet e tij qe kërkojnë zgjidhje janë po aq komplekse. Prandaj, ndërsa kërkojmë të japim një interpretim ose studim të problemeve komplekse të zhvillimit urban, vihen re tre tendenca të qarta që së bashku përcaktojnë fushën e studimeve të kompleksitetit urban (Batty & Marshall, 2011):

- Qasja sasiore, e cila përdor modelimin dhe teknikat që rrjedhin nga puna kërkimore dhe simulimet kompjuterike (ndonjëherë të quajtura *Teori e Kompleksitetit të Qyteteve*);
- Qasja cilësore që thekson filozofitë e zhvillimit të shkencës, duke përdorur qasje ndërdisiplinare dhe analoge të kompleksitetit të tilla si qëndrueshmëria, ndryshimet ose përshtatja e klimës;
- Nocioni që përkufizon optimizmin e një sistemi në një mënyrë reduktuese, nga lart-poshtë, nuk është më 'një metaforë e përshtatshme për zgjidhjen e problemeve njerëzore', e cila shfaq paradoksin e dukshëm të 'planifikimit për kompleksitetin'.

Kompleksiteti shpesh merr një formë hierarkie, kështu që një sistem kompleks konsiston në një seri nënsistemesh të ndërlidhura, që dhe ato vetë përbëhen nga nënsisteme të tjera, e kështu me radhë deri kur arrihet niveli i përbërësve elementarë (Krönert, Volk, & Steinhardt, 2001). Teoria e hierarkisë aplikon hierarki për të organizuar konceptet dhe interpretuar kompleksitetet e ndryshme. Ajo ekzaminon me detaje problematikat e shkallës, niveleve të organizimit, niveleve te vëzhgimit, niveleve të shpjegimit në një sistem kompleks të karakterizuar nga struktura dhe aktivitetet hierarkike përgjatë niveleve.

Një nga prirjet e para janë teoritë e kompleksitetit të qyteteve (CTC[16]), të cilat gjithnjë e më shumë po zhvillohen dhe studiohen nga një numër i gjerë studiuesish, që nga matematikanët, fizikanët, gjeografët, planifikuesit urban dhe projektuesit e deri te sociologët dhe ekonomistët. Juval Portugali (2011) parashtron se, gjatë tre dekadave të mëparshme, CTC ka theksuar tre arritje kryesore:

- E para, kompleksiteti i qytetit përbën tashmë një bazë të veçuar teorike për fenomene të ndryshme urbane që janë interpretuar në bazë të referencave të ndryshme teorike si për

[16] Teoria e kompleksitetit të qyteteve (Complexity Theory of Cities)

shembull planifikimi i përdorimit të tokës, shpërndarja në rang madhësie, zoonimet hapësinore, teoria e qendërsisë dhe hartëzimi njohës;

- Së dyti, CTC ka rritur depërtimin dhe kuptimin e qyteteve përmes vetive bazë të kompleksitetit, për shembull, duke demonstruar aftësinë e agjentëve të vetëm apo të veprimeve individuale që influencojnë në sistemin urban, apo se si qytetet në vetvete, përmes emergjencave mund të konsiderohen forca socio-ekonomike dhe kulturore që skalisin zhvillimin e tyre;

- Së treti, teoritë e kompleksitetit të qyteteve demonstrojnë se rendet dhe modelet e forta urbane mund të perceptohen në dukje kaotike dhe të larmishme.

Pavarësisht këtyre arritjeve, kuptimi mbi sistemet urbane, si sisteme komplekse adaptive, është shumë i kufizuar. Juval Portugali (2011) gjithashtu argumenton që përdorimi i kompleksitetit për studime urbane është jo i plotë sepse CTC kryesisht përqendrohet në modelimin e fenomeneve afatshkurtra në sisteme të thjeshta, të mbyllura që mund të studiohen duke përdorur metoda sasiore-statistikore, dhe se ata shpesh nuk arrijnë të interpretojnë domethënien e analogjive të modeluara. Prandaj, pavarësisht demonstrimit të komplekstietit si një sistem për studimet urbane, aplikimet dhe modelet e tij kuantitative janë ende në limite relative të përdorimit, dhe adresimi i zhvillimit kompleks urban është ende larg qartësimit.

Nëse i kthehemi qasjes cilësore të kompleksitetin urban, duhet theksuar nevoja për të adoptuar një botëkuptim kompleks dinamik, duke përdorur metodat e kërkimit ndërdisiplinare dhe përshkrimi i ndryshimit për analogji në sistemet social dhe ekologjike.

2.3 Ripërtëritja adaptive

2.3.1 Fazat e ripërtëritjes adaptive

Një sistem kalon në një fazë kritike kur nën-komponentët e këtij sistemi janë në ndërveprim me njëri-tjetrin (Miller & Page, 2007). Megjithatë, këta nën-komponentë nuk janë tregues të veçorive që mund të ketë një sistem, dhe njëkohësisht veçoritë e një sistemi nuk mund të deduktohen vetëm nga ekzaminimi dhe ndërveprimet mes këtyre nën-komponentëve (Aziz-Alaoui & Bertelle, 2012). Me fjalë të tjera, një sistem nuk mund të shkëputet veçmas, të verifikohet marrëdhënia mes komponentëve e më pas këto bashkohen për të kuptuar si funksionon një sistem. Fenomenet në fazë kritike janë karakteristika jolineare të një sistemi, siç mund të jenë katastrofat, pragjet dhe aftësia vetë-organizuese.

Aftësia vetë-organizuese është një fenomen i fazës kritike që ndodh kur një sistem funksionon në mënyrë të atillë që të funksionojnë "më mire" ose në mënyrë më të qëndrueshme, pa influence nga faktorë të jashtëm. Për shembull, ndërhyrjet e *"urbanizmit taktik"* janë të shtjelluar teorikisht në lidhje me kompleksitetin dhe aftësitë vetë-organizuese (Silva, 2016).

Një faze tjetër e ripërtëritjes adaptive është *feedback*[17]-u që merr një sistem. *Feedback*-u ndodh kur një rezultat i një sistemi, kthehet në po atë sistem si një rezultat hyrës[18]. *Feedback*-u negativ e zbeh shkallën e ndryshueshmërisë dhe e shtyn atë drejt një gjendjeje më të qëndrueshme. *Feedback*-u pozitiv rrit shkallën e ndryshueshmërisë, si efekt vetë-përforcues. Për më tepër, strukturat në shkallë më të gjerë mund të vijnë si pasojë e ndërveprimit mes nën-komponentëve në shkallë më të vogël dhe më pas, në të ardhmen, të ndikojnë në sjelljen e nën-komponentëve nëpërmjet kryqëzimit të shkallës.

Ripërtëritja adaptive dhe qëndrueshmëria janë tipare komplekse të sistemit adaptues që lidhen në mënyrë të drejtpërdrejtë me vetë-organizimin, *feedback*-un dhe jolinearitetitn (Holling, 1973). Sipas Walker (2004), ripërtëritja adaptive përkufizohet si "*kapaciteti i një sistemi të përthithë problematikat dhe të vetë-organizohet ndërkohë që i nënshtrohet ndryshimit në mënyrë që të ruajë ende të njëjtin funksion, strukture identitet"*. Në këtë mënyrë, një sistem i ripërtëritjes adaptive është një sistem i cili kthehet në gjendjen e tij të mëparshme pa vështirësi. Qëndrueshmëria, nga ana tjetër tenton t'i referohet specifikisht një variabël ose karakteristike të sistemit, pavarësisht paqëndrueshmërisë mes disa prej komponenteve të këtij sistemi (Aligica, 2014).

Vetë-organizimi kritik përshkruan një sistem i cili ka një pikë kritike si tërheqëse të tij e cila evoluon vazhdimisht deri sa arrin në pikën e tranzicionit dhe fazës kritike (Bak, Tang, & Wiesenfeld, 1987). Në pikën kritike, nën-komponentët e sistemit janë të lidhur drejtpërdrejte dhe ndikojnë fuqishëm në njëri-tjetrin. Këtu, edhe një ndryshim i vogël në një nën-komponent të vetëm është i aftë të prodhojë efekte të mëdha që krijojnë një efekt zinxhir në të gjithë sistemin. Shembulli më i mirë për të shpjeguar këtë fenomen është modeli i grumbullimit të rërës, siç e përshkruan edhe (Bak, Tang, & Wiesenfeld, 1987). Sipas tij, mbi një grumbull me rërë bien vazhdimisht kokrriza rërë. Më pas, ky grumbull rëre evoluon me një kënd të caktuar dhe bie në një gjendje qetësie, pavarësisht ortekëve të vogla që mund të ndodhin - kjo është pika kritike. Në këtë pikë, një kokërr e vetme rëre tjetër mund të shkaktojë në çdo moment një ortek masiv. Pas ortekut, sistemi rikthehet përsëri në gjendjen kritike dhe hyn në një gjendje ciklike vetë-përsëritëse. Në fakt, (Batty & Xie, 1999) argumentojnë se qytetet shfaqin elemente të vetë-organizimit kritik, ndërkohë që format e tyre urbane evoluojnë në mënyrë më diskrete me kalimin e kohës përmes tranzicionit.

2.3.2 Matjet e kompleksitetit të formës urbane

Kompleksitetit i mungon një përkufizim apo një mënyrë matjeje specifike dhe shpesh mbështetet në atë që e kthen në një fenomen, sjellje ose model kompleks në një kontekst të përcaktuar. Shiner et al. *(1999)* japin një përkufizim për këtë fenomen në tre formulime të ndryshme: i pari, i bazuar në entropinë, e cila është në nivele më të larta kur objektet nën studim janë të përzier dhe kanë diversitet mes tyre. I dyti, balancon shumëllojshmërinë dhe strukturën, dhe përputhet me përkufizimet tradicionale të sistemeve komplekse adaptive. I treti, ka të bëjë me rregullin, vetë-organizimin dhe emergjencat, nga të cilat struktura del nga një çrregullimi i mëparshëm.

[17] produkti ose reagimi i një sistemi
[18] Input

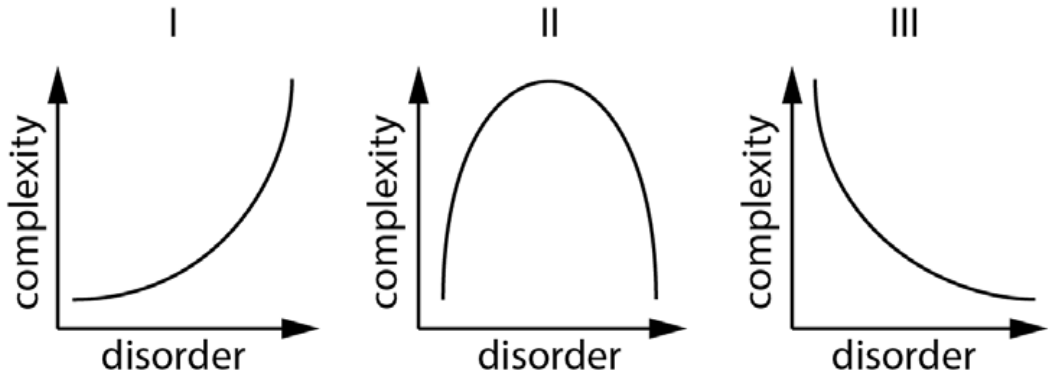


Figure 11 Tre lloje të kategorive të kompleksitetit, Burimi: (Shiner, Davison, & Landsberg, 1999)

Një rrjet kompleks hapësinor është një rrjet që ka një strukturë jo të parëndësishme. Me fjalë të tjera, struktura dhe organizimi i tij nuk është as plotësisht i rregullt dhe as plotësisht i rastësishëm. Duke iu rikthyer tre kategorive të kompleksitetit të diskutuara më sipër, kategoria e parë, maksimizon çrregullimin dhe entropinë; kategoria e dytë, maksimizon ekuilibrin midis strukturës dhe larmishmërisë dhe kategoria e tretë maksimizon rregullin (figura 11).

Forma urbane që rrjedh nga një proces planifikimi urban, është e skalitur në hapësirë dhe mund të karakterizohet nga matje të ndryshme hapësinore të kompleksitetit. Këto masa vlerësojnë karakterin e modeleve hapësinore të sistemit, duke u përqendruar në pamjet e çastit (në një kohë të caktuar, deri diku të rastësishme) në vend që të vlerësojnë dinamikën me kalimin e kohës. Modelet fizike përbëjnë gjithashtu formën urbane dhe mund të ekzaminohen në aspektin e analizave të rrjetit, strukturës fraktale, larmishmërisë (të llojeve të ndryshme) dhe entropisë së informacionit.

Entropia sipas Shannon (1948) është përdorur për të matur kompleksitetin urban (Batty M. , 2005) dhe fitimi mesatar i informacionit është përdorur për të matur kompleksitetin hapësinor të një ekosistemi (Proulx & Parrott, 2008). Teoria e Shannon (1948) mbi entropinë e informacionit ka të bëjë me sasinë mesatare të informacionit që gjendet në zbulimin e një mesazhi ose ngjarjeje. Entropia e Shannon (1948) tregon se sa më shumë lloje të ndryshme të elementeve të ketë në studim dhe sa më i barabartë dhe proporcional të jetë raporti mes tyre, aq më pak i parashikueshëm do të jetë lloji i çdo elementi i marrë në veçanti. Kjo mund të zbatohet për mesazhet abstrakte, seritë kohore ose larmishmërinë hapësinore.

Të dhëna nga studime dhe analiza të shkallëve të ndryshme urbane, duke nisur nga rajonet, tek lagjet e deri tek ndërtesat dëshmojnë rëndësinë e kritereve të vlerësimit të moduleve që përzgjidhen për analizë të thelluar. Larmishmëria si edhe mungesa e rendit apo rregullit që karakterizon qytetet sot, rrit padyshim vështirësinë e matjeve objektive. Studimi i mirëfilltë i matjeve të kompleksitetit në ambientin e ndërtuar duhet të marrë në konsideratë aspekte të ndryshme gjatë përzgjedhjes së kampionëve. Arritja e një ekuilibri ndërmjet strukturës dhe rrëmujës urbane duhet të jetë në fokusin e vlerësimit për të arritur matje dhe të dhëna të sakta. Duke qenë se qytetet janë sisteme mjaft komplekse, nga njëra anë ekzaminimi i një kampioni lagjeje mund të arrihet më së miri nëse fokusi është në trajtimin e kësaj lagjeje si një ekosistem urban. Dinamikat e jetës në komunitet, siç janë ndryshimet e shpeshta të popullsisë, dendësisë, punësimit, pasurisë, vëllimi i trafikut, etj. janë faktorë mjaft të rëndësishëm për tu konsideruar

dhe identifikuar. Ndërkohë, ka edhe faktorë të tjerë dinamikë e të ndryshueshëm, që nuk paraqesin diferenca të konsiderueshme të rezultateve edhe kur inputet ndryshojnë. Nga ana tjetër, një kampion lagjeje mund të ekzaminohet edhe si një produkt i sjelljeve njerëzore, pasi sistemet urbane dhe qytetet janë në riformulim dhe adaptim të vazhdueshëm, si pasojë e sjelljeve njerëzore (Lynch, 1954).

Sipas Geoffrey D. Boeing (2017), forma urbane mund të analizohet në disa aspekte apo kategori matëse, siç janë: kompleksiteti vizual, kohor, hapësinor, si edhe strukturor (fraktal, rrjetet), të cilat shtjellohen më poshtë:

- **Kompleksiteti vizual i formës urbane**

Në përditshmëri, informacioni vizual që perceptohet nga një këmbësor që ecën në qytet, është i paktë. Por nga ana tjetër në qytet, si një mjedis dinamik dhe kompleks, individët bombardohen me sasi të mëdha informacioni. Mënyra se si ne e perceptojmë informacionin, lidhen kryesisht me veçantitë e formës urbane, si: lartësitë e ndërtesave, ndjesitë e kufizimit të hapësirës, shkalla njerëzore, transparenca, lidhjet, kompleksiteti vizual, etj.

Sipas studimit të kryer nga Ewing et al. (2013), u gjet se njerëzit preferojnë dhe përzgjedhin që ta marrin informacionin viziv në një mënyrë që i përshtatet dhe është komforte për ta. Kompleksiteti vizual i një cilësie të mirë mbetet i lidhur në mënyrë direkte me llojet e ndërtesave, detajet e dizajnit, mobilimi urban, sinjalistika, aktiviteti njerëzor, lëvizja e diellit, si edhe detajet e materialeve dhe teksturave urbane të rrugëve, trotuareve, elementeve natyrore, etj.

Ndërsa, në kërkimin e Calvante et al. (2014), u konkludua se kompleksiteti i perceptuar në hapsirën urbane në qytet, ndikon në interesin visual të këmbësorëve dhe në sjelljen e drejtimit të automjeteve. Ai propozon një metodologji për matjen objektive të kompleksitetit, duke u bazuar në kontrastin lokal dhe në frekuencat hapësinore. Ndër të tjera, ai zbulon se kompleksiteti më i lartë shoqërohet me praninë e objekteve me kontrast të lartë midis tyre dhe zona të karakterizuara nga frekuenca hapësinore më të ulëta se mesatarja në mjedis.

- **Matjet kohore të formës urbane**

Matjet që vijnë si rezultat i analizave të cilat studiojnë dinamikat dhe ndryshimet që ndodhin në prizmin e nocionit “kohë” njihen si matjet kohore. Metodat që përdoren për të vjel matje të tilla janë të shumta, por të gjitha përdorin përfshirjen e elementit kohor në hapësirë. Disa studiues, si: Lyapunov, Shannon, etj. kanë realizuar studime të cilat mund të aplikohen edhe në projektimin urban në shkallë lagjeje apo qyteti. Sipas Kuper (2017), të dhënat serike që vilen nga ky aspekt matës, pra nga koha, mund të abstraktohen për të arritur matje apo vlera që lidhen me lëvizjen njerëzore në hapësirën (hapësirë fizike, qytet, etj.) që rrjedh si pasojë e vendimeve të dizajnin urban.

- **Matjet hapësinore të formës urbane**

Matjet hapësinore të kompleksitetit të formës urbane, kur rrjedh prej planifikimit, sigurisht që janë më të unifikuara. Këto të dhëna, kanë vlerë më shumë në një moment të caktuar, sesa si

vlera dinamike në kohë. Larmishmëria, është matja hapësinore më e përdorshme e kompleksitetit, në planifikimin dhe projektimin urban.

Jane Jacobs (1961) vlerësoi se përdorime të ndryshme të tokës kanë aftësinë e tyre për të krijuar sinergji nga funksionet plotësuese. Cervero dhe Kockelman (1997) gjithashtu argumentuan se diversiteti i përdorimit të tokës është një tipar kyç që formon sjelljen e të udhëtuarit në mjediset urbane.

Salat et al. (2010) identifikojnë tre lloje të larmishmërisë apo diversitetit hapësinor të formës urbane të lidhura me kompleksitetin: larmishmëria midis objekteve të ngjashme, larmishmëria në shpërhapjen hapësinore dhe larmishmëria e shkallës. Larmishmëria midis objekteve të ngjashme i referohet karakteristikave të ndryshme midis objekteve të njëjta dhe mund të bëhet një vlerësim i drejtë, kur shpërndarjet e njëtrajtshme janë optimale. Megjithatë, ky qëndrim bëhet i rrezikshëm në nivel planifikimi qendror. Matjet hapësinore të shpërhapjes dhe të formës fizike janë të vlefshme për të karakterizuar uniformitetin, rastësinë ose kompleksitetin hapësinor të ekosistemeve, e si të tilla mund të aplikohen edhe në mjedisin e ndërtuar.

Në një sistem kompleks, përqendrimi ose grupimi I funksioneve të njëjta, mund të bëjnë që qendrat e punës të grumbullohen në zona të caktuara ((Jacobs J. , 1969) (Sevtsuk, 2014)). Në këto raste, pyjet apo hapësirat e gjelbërta janë të përqendruara e të grumbulluara dhe jo të shpërndara në hapësirën urbane. Larmishmëria e shkallës dhe niveli i observimit, do të diskutohet në kapitujt e mëposhtëm.

- **Matjet strukturore të formës urbane**

Nëse vazhdojmë debatin lidhur me larmishmërinë, ngrihen pyetjet që lidhen me matjet strukturale në nivel shkalle. Matjet strukturore lidhen me aspektin fizik të një sistemi. Ato aplikohen vazhdimisht në qytete dhe janë matjet më të përdorshme të atributeve të kompleksitetit në një qytet, pasi ato na japin të dhëna të matshme lidhur me strukturën fizike të qytetit dhe shtrirjen e tij. Densiteti në vetvete mund të jetë një përfaqësues i thjeshtë për kompleksitetin, pasi mund të përshkruajë sesi një numër më i madh aktivitetesh ndërvepron në të njëjtën zonë, duke nënkuptuar strukturën urbane dhe lidhjet midis tyre. Matjet struktore, në shkallën e projektimit urban, ndahen kryesisht në dy kategori: masat e strukturës fraktale dhe analiza e rrjetit.

2.4 Fraktali

2.4.1 Sintaksa e formës dhe rregulli fraktal

Ndryshe nga Viena apo Barcelona, Parisi dhe Tokio janë zhvilluar pa një plan të përgjithshëm për qytetin. Por, aq sa mospërfillëse mund të duket nga çdo rregull topografik, sidoqoftë, struktura e tyre dëshmon një formë shumë komplekse të rendit, e ndryshme për të dy qytetet, çfarë iu jep atyre një vulë të patjetërsueshme të identitetit. Parisi, për shembull, të kujton një mozaik gjigand, më afër planit urbanistik të Pergamon[19]-it se sa të Qytetit Radial[20] të Le

[19] Akropoli i Pergamonit ishte kryeqyteti i dinastisë helene Attalid, një qendër kryesore e të mësuarit në botën e lashtë

[20] Plani modern i Le Corbusier në vitin 1920, për Parisin

Corbusier-së. Bruno Fortier (1989) e përshkruan Parisin si *"një lloji gjigandi territorial në të cilin rrugët shtrihen mbi rrënojat e ish-manastireve, kalldrëmet kthehen në kopshte, faltoret përfshihen brenda qytetit, dhe gjurmë të Hartës së Botës që lidhin disa prej monumenteve me njëri-tjetrin, po ende të paprekura, të gjitha duke luajtur distancën e duhur, një rezultat që asnjë projekt nuk mund ti ketë bashkuar."*

Ende, ajo çfarë rezulton nga planet e Parisit dhe Tokios nuk është asnjëherë pa lidhje logjike. Planet kurrë nuk pushojnë së gjeturi strukturën e qëndrueshme në çdo shkallë, përsëri e ndryshme për të dy qytetet. Dhe shtresat e rrugëve, pavarësisht sa janë deformuar nga topografia dhe historia, shfaqin karakteristika konstante: një distancë mesatare midis kryqëzimeve rreth 120 metër për Parisin, dhe për ato qytete të projektuara nga evropianët si Hong Kong apo qendra e Melburnit, ndërkohë vetëm 50 metra në Tokio. Çdo qytet zgjedh, në mënyrën se si i adaptohet kulturës së vet dhe secilës periudhë por dhe që rezulton i qëndrueshëm përgjatë kohës, duke përdorur një numër të kufizuar zgjidhjesh, prezenca e të cilave strukturon peizazhin e tij. "*Kudo evidentohen të njëjtat figura themelore, për të treguar se çfarë ishte humbur në një vend apo kundërshtuar nga topografia e historia, ishte fituar në zhvillimin e një teksture në të cilën fati, nuk luante më një rol mos strukturimi"* e vëzhguar nga Bruno Fortier (1989).

Sipas, Salat et al. (2011), fakti që strukturat matricore dhe topologjike të këtyre qyteteve pa plan, mbesin çuditërisht të qëndrueshme nën presionin e ndryshueshmërisë ekstreme të krijimit të formave aksidentale, ngre dy pyetje:

E para konsiston nëse 'zbulimi i qytetit' është më pak fakt i investimeve në projekte të veçanta se sa "në gjetjen e rregullave të formësimit dhe bashkekzistencës së një jetese, si një fushë e hapur e elementëve në mënyrë konstante". Kompleksiteti i këtyre rregullave të formimit, ka humbur sot në varfërimin e shkaktuar nga modernizmi, i cili e redukton qytetin në objekte të izoluara. Stoku i "formave ideale" të qyteteve tradicionale, ka koherencën e vet që strukturon rolin e hapësirave boshe dhe plot, thyerjeve dhe vazhdimësive, rendeve dhe pamjeve. Modernizmi na ka trashëguar "anti-forma" të pastrukturuara që dështojnë në koherencën e qytetit apo në mundësinë e riprezantimit të tij.

Dhe kjo hap një nivel të dytë studimi, i cili zhvillohet gjerësisht, çfarë përfshin kuptimin e minimumit të pragut të kompleksitetit dhe artikulimit, në të cilat këto rregulla të organizimit të kompleksiteteve urbane mund të krijojnë një gjuhë të kuptueshme, dhe të prodhojnë një mjedis njerëzor dhe jo një humbje shkatërrimtare të qytetit. Ne duhet të shohim për rregullat e organizimit të këtyre zonave urbane, jo të kopjojmë të shkuarën, thënë më mirë të lëvizim drejt morfologjive të brendshme dhe integrimit të shkallëve të pazbuluara më parë. Për të analizuar drejt hapësirat, format urbane duhet të shohim në një tjetër ekuilibër, atë të rilidhjes së njerëzve përgjatë një serie shkallësh të njëpasnjëshme, duke rizbuluar rregullin e humbur fraktal të qyteteve historike.

2.4.2 Universialiteti i fraktaleve dhe pikëpamjet teorike mbi to

Ne fillimisht duhet të kuptojmë se si çështjet "e gjalla" dhe "jo të gjalla" janë organizuar në një mënyrë komplekse për të formuar të tërën. Fizika kuantike, teoria e kompleksitetit, sistemet e analizave, inteligjenca artificiale dhe gjeometria fraktale konvertojnë për të na siguruar një pamje shumë më komplekse të universit dhe jetës. Gjeometria e qyteteve historike shfaq ketë shkallë të lartë të kompleksitetit. Universialiteti i ligjeve të fizikës na lejon të japim një përshkrim të strukturave të përbashkëta të qytetit të fshehura nën ndryshimin e sipërfaqes së fakteve urbane. Kjo është ajo çfarë Bernard Sapoval[21] quan "universialiteti i tretë". Që nga viti 1970, zhvillimet shkencore u ndryshuan, me dy qëndrime të reja:

(i) Njohja e ngjashmërive në sjelljen e sistemeve, edhe kur drejtohen nga ndërveprime të brendshme të ndryshme.
(ii) Nevoja emergjente e një gjuhe të re, ajo e gjeometrisë fraktale, që përshkruan zhvillimin e një objekti sipas një simetrie të veçantë, objekte pjesë të të cilave i përngjajnë të tërës, ku i vetmi variant i të cilit është zgjerimi apo zvogëlimi. Qytetet, të paktën qytetet historike, janë midis sistemit "universal" që zotëron këtë simetri të zgjerimit dhe ngjashmërisë së vetvetes në shkallë të ndryshme.

Përdorimi i gjeometrisë fraktale si një mjet për klasifikimin e fenomeneve urbane, është një hap i madh në avancimin e kuptimit të kompleksitetit të tyre. Gjeometria fraktale zbulon një "rend të fshehtë" në morfologjinë e strukturave urbane, duke zbuluar strukturat e brendshme hapësinore dhe organizimet në nivele të kompleksitetit në shkallë të ndryshme që nuk mund të shihen duke përdorur rrjetën analitike të gjeometrisë së zakonshme. Gjeometria fraktale gjithashtu na lejon të ndërtojmë modele shpjeguese për morfogjenezën e strukturave urbane brenda strukturave fraktale.

Qytetet historike janë një laborator për të ekzaminuar marrëdhënien midis njerëzve, klimës dhe mjedisit urban. Duke u përballur me forcat e natyrës: tokën, diellin, erën, qytetet fraktale ishin rezultat i përpjekjeve për mbijetesë të brezave në kohëra. Qytetet historike duhet të shërbejnë si streha, në të cilat koncepti i qyteteve duhet të formohet. Anasjelltas, planifikuesit e qyteteve moderne, u nisën për të shkretuar qytetet e vërteta, në emër të parimeve planifikuese dhe teorive rreth përparësive të rrjeteve ortogonale, ndërkohë kur të gjithë qytetet historike janë të kompleksuara nga topografia dhe hidrografia e tyre e çrregullt, dhe nga rrugët e lakuara të krijuara nga njerëzit.

Për të kuptuar kompleksitetin e qyteteve duhet bërë një analizë e shkallëve të tyre, duke zbuluar nivelet hierarkike të organizimit të tyre. Në këto hierarki, disa grupe të njëpasnjëshme nivelesh shfaqin një rregull shumë më të mirë se të tjerët. Përshkrimi i një tërësie "të mirë-organizuar" përgjithësisht prezanton nocionin e strukturës: elementi i rendit më të lartë është i përfshirë brenda elementëve të rendit të ulët sipas një skeme të mirë-përcaktuar. Rendi hierarkik që lidh frekuencën e paraqitjes së elementeve me madhësinë e tyre është, siç do ta shohim, është një rend fraktal.

[21] Fizikant francez që ka dhëne kontribut në studimin e fraktaleve.

Termi është një neologjizëm i shpikur nga Benoit Mandelbrot (1983) nga latinisht '*fractus*'[22], e derivuar nga folja '*frangere'*. Fraktal do të thotë i fragmentizuar, i ndarë, i çrregullt, i ndërprerë. Nëse flasim përgjithësisht, teoria fraktale është një teori e lidhur me thyerjen, copëzimin, shpërndarjen, porozitetin, koklavitjen. Por, rëndësia e teorisë është të identifikojë një rend nën pamjen e çrregullt të formave: rendi kompleks i objekteve i pështjelluar në mënyra të shumëfishta. Gjenerimi i fraktaleve matematikore, nga përsëritja e funksioneve krijon kompleksitetin jashtë rregullave të thjeshta. Më shumë se sa një teori e formave të çrregullta, kjo është një teori e indekseve të rregullta të rastësisë. Në prezantimin e tij të gjeometrisë fraktale, Benoit Mandelbrot (1983) përshkruan rëndësinë e saj me termat që vijojnë: *"Retë nuk janë sfera, malet nuk janë konike, vijat bregdetare nuk janë rrethore dhe lëvorja nuk është e lëmuar, po ashtu as drita nuk përshkon në vijë të drejtë. (...) Natyra nuk shfaq thjesht një shkallë më të lartë por një nivel krejtësisht të ndryshëm të kompleksitetit (...) Ekzistenca e këtyre modeleve na sfidon të studiojmë këto forma që euklidi i lë mënjanë si të jenë "të paformë"."*

Një objekt i çrregullt nuk është as i paorganizuar dhe as kaotik. Gjeometria fraktale na jep një krahasim më të mirë të impaktit të vendimeve njerëzore mbi të ardhmen e qyteteve dhe na lejon të gjenerojmë tregues të rinj për të matur progresin në zhvillimin e qyteteve të qëndrueshëm. Qyteti i shfaq strukturat në të gjitha shkallët. Rrjedhimisht ai është fraktal në kuptimin më të gjerë të termit. Karakteri kompleks jo linear i morfogjenezës së tij, duke arritur kulmin në strukturën fraktale shpjegon këtë heterogjenitet të strukturës urbane. Projektuesit modern ngatërruan rendin vizual gjeometrik për mirëfunksionimin e rendit shoqëror dhe të qëndrueshëm në mjedisin e ndërtuar (Roy, 2005). Studiuesit që ndjekin pas Jane Jacobs (1995), kanë argumentuar se projektimi urban i thjeshtuar me një qëllim të vetëm shkatërron kapacitetin funksional dhe energjinë e brendshme të tij.

Gjeografia urbane dhe në veçanti teoria e përqendrimit të vendbanimeve përfshin faktin që qytetet ekzistojnë jo të veçuara, por si pjese e sistemeve hierarkike që Batty dhe Longley (1994) i përcaktojnë si bindje në rend dhe madhësi, sipas një ligji të shpërndarjes fraktale. Ai pohon se shkalla e observimit mbetet një mjet analitik i diskutueshëm dhe jo i vërtetë, sepse e sheh analizimin e fakteve urbane të pavarur nga shkalla e observimit. Ai gjithashtu shkon më tej duke bërë një analizë krahasimore të konfigurimit fizik të qytetit bazuar në dy procese kryesore: ai i kontrolluar me orientim nga "sipër-poshtë", e që prodhojnë struktura formale drejt gjeometrisë ortogonale, dhe ai nga "poshtë-sipër" që si proces tipologjik formues, që tenton të prodhoje struktura të çrregullta formale. Diferenca vërehet qartë mes formave të urbanizmit modern dhe atij vernakular, një qasje analitike që merret në shqyrtim nga disa teoricienë.

Gjithashtu, Batty (2005), pohon se duhet një analizë e plotë, e të dy dimensioneve, si masës fizike ashtu edhe flukseve. Në një shkrim teorik që tenton të ngrejë një "shkence" të qytetit ai pohon gjithashtu rëndësinë e rrjeteve dhe flukseve urbane.

Në këtë studim, koncepti i fraktaleve do të lidhet me rrjetet dhe do të zhvillohet vetëm në nivel teorik. Fraktali shpërndahet sipas ligjit të fuqisë. Siç u përmend edhe më sipër, ka një numër të vogël elementësh më të mëdhenj, numër mesatar elementësh mesatarë dhe shumë elementë më

[22] I thyer.

madhësi të vegjël. P.sh. nëse do shikojmë një rrjet rrugor në një qytet, në një shkallë të madhe, do të vërejmë arteriet kryesore dhe bulevardet e mëdha; nëse afrohemi më tepër, do të vërejmë rrjet më të dendur dhe rrugët kryesore; por nëse tentojmë më shumë të afrohemi do të shohim një rrjet shumë të dendur dhe rrugë (figura 12). Dimensioni fraktal, D, është një matje statistikore, sesi kompleksiteti i formës urbane ndryshon, në varësi të shkallës që bëhet matja.

$$D = \log N/\log S,$$

Ku 'D' është vlera e dimensionit fraktal, 'N' është numri i objekteve që gjenerohen në një shkallë të caktuar dhe 'S' është faktori shkallë

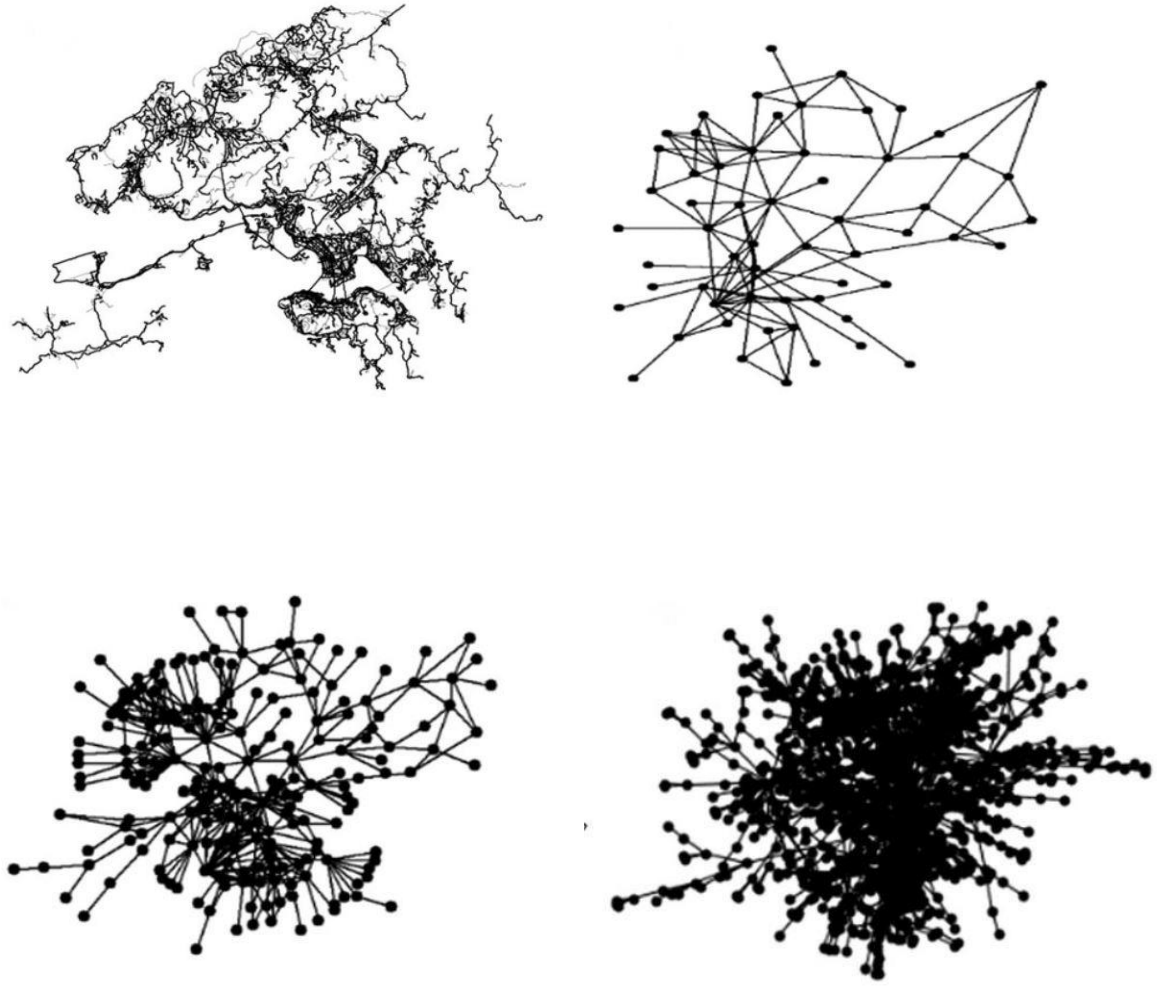


Figure 12 Karakteri fraktal i një rrjeti, parë në shkallë të ndryshme observimi. Burimi: (Lan, Li, & Zhang, 2019)

2.5 Rrjetet

2.5.1 Parantezë mbi rrjetet dhe lidhshmërinë e tyre

Lidhshmëria është një nga tiparet themelore në krijimin e një shoqërie, ku shkatërrimi i saj, mund të shkatërrojë shoqërinë në vetvete dhe origjinën e saj. Lidhjet, që krijohen në një qytet,

janë zemra dhe vetë thelbi i qytetit, sa i përket përdorimit të përditshëm dhe tendencave më të përgjithshme në përvetësimin e hapësirës. Artikulimi i qytetit tek ata që e jetojnë në përditshmëri, ndryshon thellësisht nga praktika që urbanistët promovojnë. Qytetet e zhvillojnë formën e tyre në kohë dhe në hapësirë. Më tepër sesa si hapësira inxhinierike të projektuara, ato janë hapësirë njerëzore dhe studimi i formës së qytetit, zbulon funksionin antropologjik të tij, duke qendërzuar qeniet njerëzore si krijesa të gjalla me aftësinë e të folurit. Të studiosh mbi lidhshmërinë e një qyteti, do të thotë të kuptosh në mënyrë efikase kompleksitetin e tij. Lidhshmëria nxit zhvillimin e të gjitha shoqërive njerëzore. Organizimi i shoqërisë varet nga mënyra sesi njerëzit ndërveprojnë në jetën e përditshme të tyre. Ky ndërveprim, në masën më të madhe orientohet nga rrjetet rrugore (Lynch, 1960).

Ekziston një ndryshim ndërmjet rrjetit abstrakt që projektohet dhe atij që në vërtetë njerëzit lëvizin. Kevin Lynch (1960) dhe Bill Hillier (1996), të dy theksuan rëndësinë e lidhshmërisë vizuale për qëllime orientimi dhe për të krijuar një imazh të lexueshëm të qytetit. Por lidhshmëria vizuale nuk përputhet me lidhshmërinë e njerëzve në hapësirë. Rrjetet urbane të projektuara nga planifikuesit, janë krijuar në hapësirë duke ndjekur rregullat e formave simetrike, të ngjashmërisë apo formave të ndërmjetme, ndryshe nga ato të krijuara prej njerëzve në kohë dhe në hapësirë, të cilat janë më fluide. Struktura e këtyre rrjeteve, në qytetet historike, ka kompleksitet të lartë dhe reduktimi në forma më të thjeshta është shumë i vështirë. Në qytetet moderne me rrjet të rregullt, është e pamundur të rrisësh lidhshmërinë e tyre. Nikos Salingaros thekson se sipas teoremës bazë të matematikës, dy pika bashkohen me një drejtëz të vetme, por ai shton se ato mund të bashkohen me një numër të pafundmë vijash të lakuara. Rrjetet urbane të qyteteve mesjetare kanë një numër të pafundmë lidhjesh të tilla. Për mijëra vite njerëzit i kanë ndërtuar qytetet e tyre me rrjete të lakuara dhe shpjegimi për to është shumë i thjeshtë. Rrjetet e drejta dobësojnë lidhshmërinë dhe hierarkinë fraktale, të dobishme për jetën urbane.

Tre principet bazë të strukturës së rrjeteve

Urbanizmi modern përdori dhe abuzoi me format urbane, për të krijuar hapësira të cilat kur shiheshin nga lart dukeshin të rregullta dhe me lidhshmëri të lartë, ndërkohë ato ishin jashtë shkallës njerëzore dhe nuk garantonin lidhshmëri. Sipas Salingaros (1998), tre principe bazë gjenerojnë rrjetin urban: 1) *nyjet*, të cilat përshkruhen si pika ku gjenerohen aktivitetet njerëzore. Nyjet përfshijnë shtëpitë, vendin e punës, dyqanet, kishat etj. Elementët natyrore dhe arkitektonikë përcaktojnë kufijtë e këtyre nyjeve dhe itenerarët lidhës të tyre. 2) *Lidhjet* (skajet) Lidhjet ndërmjet nyjeve plotësuese me njëra tjetrën dhe jo atyre të ngjashme. Për të rritur numrin e lidhjeve ndërmjet dy pikave, disa prej lidhjeve duhet të jenë të kurbTA ose të çrregullta ose rrjeti duhet të përmbajë një numër të madh laqesh. 3)*hierarkia*, një rrjet urban i suksesshëm është i strukturuar nga një hierarki fraktale e rregullt e lidhjeve në shkallë të ndryshme. Kjo siguron lidhshmëri, në forma të ndryshme pa qenë kaotike. Në përfundim, Salingaros (1998) thekson se lidhshmëria ekziston në shkallë të ndryshme të qytetit dhe nuk është e thënë që ato të provohen matematikisht. Qytetet që manifestojnë kompleksitet të lartë, kanë shkallë të lartë të lidhshmërisë së tyre. Ato janë si truri, që edhe nëse një ose disa prej lidhjeve prishen ato ende funksionojnë.

2.5.2 Këndvështrim teorik mbi analizat e rrjeteve

Rrjeti i rrugëve në një qytet është shtylla kurrizore e tij. Konfigurimi i strukturës së tij, ndikon në mënyrë të drejtpërdrejtë formën fizike të qytetit. Për shkak të arsyeve të ndryshme, si zhvillimit historik, planifikimit urban, elementëve natyrorë, mjedisit social kulturor, struktura e rrjeteve të një qyteti mund të ndryshojë nga një qytet në tjetrin. Këndvështrimet e studiuesve, lidhur me analizimet e rrjeteve, janë për qëllime të ndryshme, si p.sh.: për sigurinë rrugore (Marshall & Garrick, 2010) për mbrojtjen ndaj zjarrit (Usui & Asami, 2011), për shëndetin public (Marshall, Piatkowski, & Garrick, 2014), për përdorimin e energjisë (Mohajeri, Gudmundsson, & French, 2015), për krimet (Summers & Johnson, 2016). Nevoja për leximin e strukturës së rrjeteve, nuk lidhet vetëm për të kuptuar formimin dhe zhvillimin e qyteteve të ndryshme, por edhe për të hulumtuar sesi struktura të ndryshme rrjetesh ndikojnë në shoqërinë tonë, në ekonomi dhe në mjedisin e ndërtuar.

Nëse shohim në kohë studimet kanë pasur orientime të ndryshme në zhvillimin e koncepteve të rrjeteve. Kështu Marshall (2005) përcaktoi pesë tipa të rrjeteve: lineare, si pemë, rrezore, qelizore dhe hibride (figura 13).

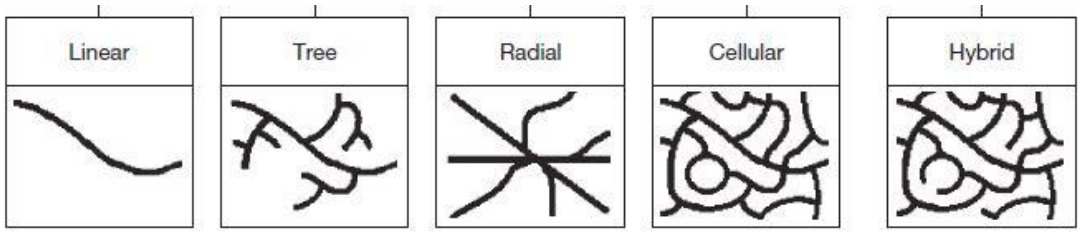


Figure 13 Tipet e rrjeteve. Burimi: Marshall (2005)

Studiues të tjerë theksuan unazat (Heinzle, Anders, & Sester, 2006) dhe strukturën rrjetëzuese (Visscher, Yang, & Goddard, 2010) . Shumica e studimeve analizojnë modele të ndryshme brenda rrjetit. Studiues si Porta (Porta, Vito, & Latora, 2006) kanë trajtuar rrjetet në aspektin strukturor, si p.sh. kryqëzimet e rëndësishme dhe segmentet në një rrjet, por qasje e tyre janë përdorur shumë rrallë për të analizuar konfigurimin e një rrjeti. Ka studime të tjera të cilat përdorin konceptin e entropisë (Shannon, 1948) për të matur modelin e rendit ose çrregullimit, të një rrjeti (Gudmundsson & Mohajeri, 2013). Entropia orientuese, e cila mat orientimin e rrugëve në një qytet, është parë si një tregues i rëndësishëm i modelit të rrjetit dhe është e lidhur me leximin e formës urbane (Boeing G. , Urban spatial order: street network orientation, configuration, and entropy, 2019). Ai përdori entropinë orientuese për të ekzaminuar qytete në të gjithë botën dhe gjeti se qytetet e SHBA-së dhe Kanadasë kishin ngjashmëri të mëdha midis tyre.

Një qasje tjetër e studiuesve lidhet me marrëdhënien që krijohet ndërmjet rrjeteve rrugore dhe karakteristikave natyrore ku ngrihet ai qytet, si luginat, malet dhe kodrat, liqenet dhe detet (Mohajeri & Gudmundsson, 2014). Për shembull, Mohajeri (2013) hulumtoi modelin e rrjetit të rrugëve në qytetin e Khorramabad (Iran) dhe lidhja e tij me malet dhe kodrat. Gjithashtu, Mohajeri (2013) konstatoi se rrjeti rrugor në qytetet braziliane është i lidhur gjeometrikisht me vijën bregdetare. Konkretisht, sa më e lakuar të jetë vija bregdetare, aq më e madhe është shpërndarja në orientimin e rrugëve, dhe aq më e madhe është entropia e lidhur me të.

Kohët e fundit, studimet i kanë kushtuar vëmendje edhe analizës së rrjeteve në një shkallë më të madhe, në nivel kombëtar ose edhe global. Boeing (2020), propozoi analizimin e rrjeteve të rrugëve me madhësi të mëdha të kampionit, nëpërmjet përdorimit të një softi të hapur. Boeing (2020), mati evoluimin e rrjeteve të rrugëve në të gjithë SHBA. Barrington-Leigh dhe Millard-Ball (2015), kanë prezantuar një analizë të shtrirjes së rrjetit të rrugëve, e cila mund të zhvillohet në të gjithë botën. Edhe pse është bërë më e thjeshte, për shkak të të dhënave masive që lidhen me zhvillimet teknologjike, shumë pak studime kanë arritur të analizojnë në mënyrë sasiore ngjashmëritë apo ndryshimet e treguesve për rrjete të ndryshme në vende të ndryshme.

2.5.3 Analizimi teorik i rrjeteve sipas Stephen Marshall

Si një qasje fillestare, ne mund të analizojmë studimin e rrjeteve të rrugëve sipas rritjes së tyre historike siç bëri Stephen Marshall, duke propozuar një tipologji A, B, C, D.

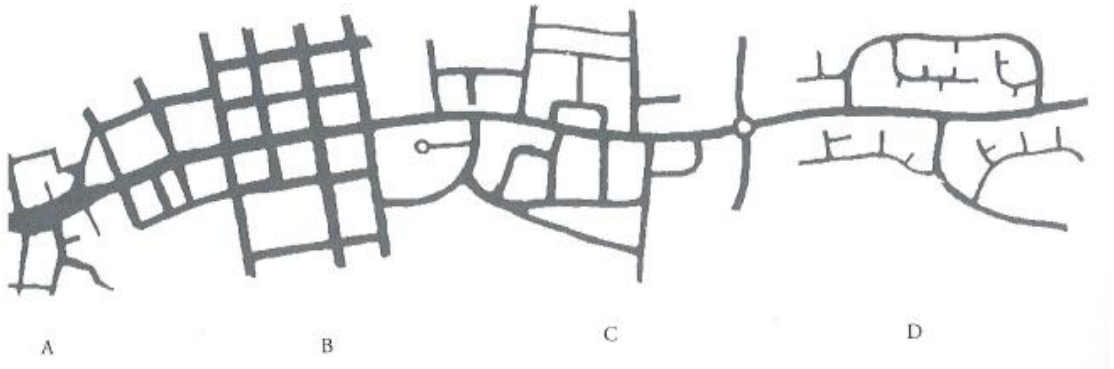


Figure 14 Evoluimi i rrjeteve nga qendra në periferi të qyteteve. Burimi: Marshall (2005)

Lloji A korrespondon me qytetet e vjetra, të karakterizuara nga rrugë të lakuara dhe me orientime të shpërndara. Këto janë veçanërisht të zakonshme në qytetet të rrethuara me mure, në ato italiane dhe holandeze. Ky lloj u përhap në shumë vende në një periudhë të shkurtër. Si rezultat, ai është edhe shumë i larmishëm dhe i qëndrueshëm. Pëlhurat organike bazohen në logjikën e *qendërsisë* dhe kufizimeve të vet situatës urbane. Shpesh, rrjeti përkthen në mënyra hapësinore funksionin bazë të qytetit: shkëmbimin e të mirave dhe rrjedhimisht qendërsia. "Maksimizimi i shkëmbimeve i kombinuar me minimizimin e udhëtimeve imponon një konvergjencë të flukseve dhe arterieve kryesore që vijnë nga jashtë qytetit dhe takohen në një pikë qendrore. Kjo mund të jetë një port ose një urë, ose një shesh qendror dhe ndërtesat e tij fetare ose tregtare. Vendosja bashkëqendrore e rrugëve mbështjellëse është dytësore. Ato plotësojnë konfigurimin për të përmirësuar lidhjen (Marshall S. , 2005).

Në këtë tip rrjeti, hapësira përfaqëson autenticitetin e saj. Topografia dhe hidrografia mund përforcojnë ose shqetësojnë parimin e qendërsisë. Në Romë, konvergjenca e luginave i jep vetes një strukturë radiale që përforcon mënyrën e shtrirjes së rrugëve. Dualiteti qytet i lartë - qytet i ulët, reflektohet edhe në strukturën e këtyre rrjeteve. Në zonat e larta, rrugët janë më të vjetra, më të ngushta dhe më komplekse; në zonat e ulëta, ku hapësira është më e madhe, ajo është më e gjerë dhe e rregullt. Në strukturat urbane organike, përbërësit ndërthuren në një rrjet kompleks dhe kanë lidhshmëri të lartë. Në këto qytete jo vetëm që është e mundur ecja, por shpesh janë edhe efikase për udhëtimet me automjet, sepse sistemi ofron shumë rrugë të mundshme midis dy pikave. Dhe duke qenë se i gjithë rrjeti është i disponueshëm për të lëvizur, shpërndahet më mirë dhe trafiku mbetet i lehtë në shumicën e rrugëve.

Lloji B korrespondon me zgjerimet e planifikuara në qytetet e vjetra, duke filluar nga shekulli i XVIII-të në Evropë, apo krijimi i qyteteve kolonizuese, qoftë në antikitetin helen apo romak apo gjatë pushtimit të Amerikës. Mbizotërimi i rrugëve pingul me njëra-tjetrën i jep qytetit një karakter të dyanshëm. Kjo konstatohet, p.sh., në shtrirjen e Barit ose në atë të Barcelonës. Në këtë tip përfshihet gjithashtu lloji i rrjetit drejtkëndor që u përdor për kolonizimin në Amerikën e Veriut. Që nga koha e Hippodamus-it të Miletus[23], rrjeti ka qenë një instrument i pushtimit të tokës. Ky sistem u riprodhua në të gjitha qytetet greke. Plani i qyteteve romake i strukturuar nga dy akse: Veri-Jug dhe Lindje-Perëndim, ndikoi në shtrirjen e shumë qendrave të qyteteve antike në Evropë. Ndikimi klasik francez shënjoi disa qytete të Amerikës së Veriut të themeluara në shekullin e XVIII-të, duke përfshirë Detroitin, Mobile, New Orleans dhe Saint Louis. Por në përgjithësi, planimetria e shumicës së qyteteve amerikane janë rezultat i nënndarjeve të vendbanimeve, jo i planifikimit të qëllimshëm urban.

Tipi C është karakteristikë e strukturës qendrore të një fshati, ose e një shtrirjeje periferike përgjatë një rruge. Në këtë lloj rrjeti i rrugëve formon një përzierje të ortogonales dhe asaj organike. Ky është një tip urban më pak i dendur që megjithatë ruan karakterin e tij urban.

Së fundi, **tipi D** është i hierarkizuar dhe tipik për strukturat moderne. Shpesh karakterizohet nga rrugë të lakuara dhe rrugë pa krye. Ajo gjendet sot në shumicën e zonave periferike, veçanërisht në Shtetet e Bashkuara. Mjedisi i ndërtuar nuk jep direkt në rrugë; është i shpërndarë, me shtëpi të veçuara me kopshte. Krijimi i një rrjeti ekskluzivisht të përqendruar tek makinat ngre probleme serioze për sa i përket aksesit, densitetit dhe konsumit të energjisë. Në të vërtetë, si rrjeti ashtu edhe struktura urbane shfaqin densitet të ulët dhe lidhshmëri të vogël. Pëlhura urbane konsumon sasi të mëdha energjie dhe materialesh dhe nuk është elastike dhe adaptueshme. Një zhvillim që maksimizon transitin e makinave nuk është i zbatueshëm, aq më tepër sepse logjika që qëndron pas tij çon në mënyrë të pashmangshme, në një rreth vicioz, në një përkeqësim të situatës: sa më i përhapur të bëhet qyteti, aq më shumë përdoret makina dhe sa më shumë të përdoret makina, aq më të mëdha duhet të jenë infrastrukturat dhe, për rrjedhojë, qyteti shpërndahet edhe më shumë.

2.5.4 Paraqitjet dhe analizat e Rrjetit

Shkenca e rrjetit ka në fokus gamën e gjerë të ndërveprimeve, lidhjeve, dinamikave dhe proceseve që ndodhin brenda një sistemi. Në një kontekst urban, atributet strukturore të rrjeteve të qytetit mund të ndikojnë në mënyrën se si organizohen lidhjet fizike të një sistemi urban dhe ndikojnë ndërveprimet, lidhjet dhe dinamikat komplekse njerëzore (Baynes, 2009) (Comunian, 2011). Siç argumenton Glaeser (Glaeser, 2011), ndër të tjera, qytetet ekzistojnë për t'u lidhur njerëzit. Shkenca e rrjetit është ndërtuar mbi bazën e teorisë së grafeve, një degë e matematikës. Një graf është një paraqitje abstrakte e një grupi elementësh dhe lidhjet ndërmjet tyre (Tinkler, 1979). Elementet në mënyrë të ndërsjellë quhen kulme ose nyje, dhe lidhjet ndërmjet tyre

[23] ishte një arkitekt, urbanist, mjek, matematikan, meteorolog dhe filozof grek i lashtë, i cili konsiderohet të jetë "babai i urbanistikës evropiane", dhe për nder të tij përdoret termi "plani i Hippodamus" (plani i rrjetit) të qytetit

quhen skaje (Downey, 2012). Në këtë studim, do të përdoret termi "nyje", i lidhur me konceptin e morfologjisë urbane.

Numri i nyjeve në graf përfaqësohet zakonisht si *'n'* dhe numri i skajeve si '*m'*. Dy nyje janë ngjitur nëse i lidh një skaj, dy skajet janë ngjitur nëse ndajnë të njëjtën nyje, dhe një nyje dhe një skaj janë rastësore nëse skaji lidh nyjen me një nyje tjetër. Shkalla e një nyje është numri i skajeve që ndodhen në nyje, dhe fqinji i tij janë të gjitha ato nyje me të cilat nyja është e lidhur me skaje (figura 15).

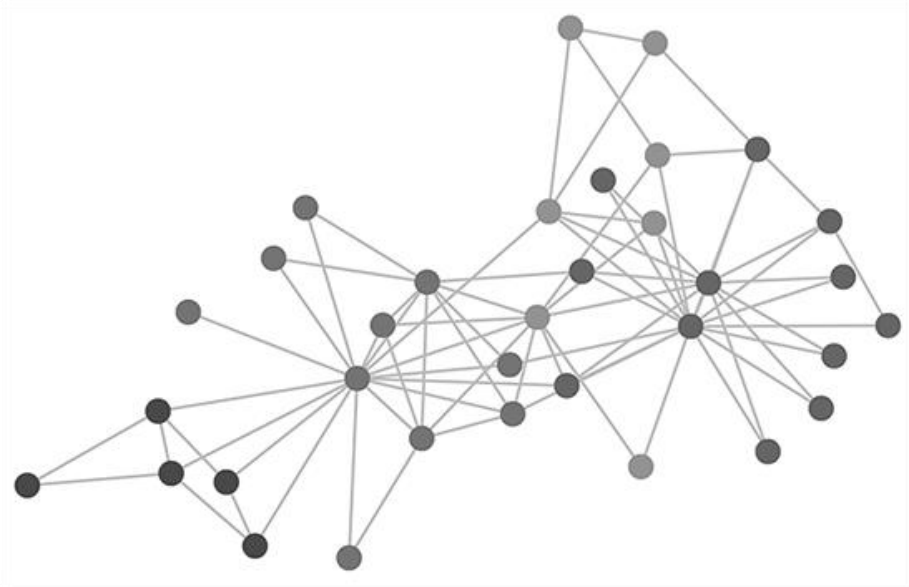


Figure 15 Prezantimi i një rrjeti, evidentimi nyje dhe skaje. Burimi: autori

Një graf i padrejtuar *'undirected graph'* i ka skajet të padrejtuara (d.m.th., çdo skaj tregon reciprokisht në të dyja drejtimet) por një graf i drejtuar *'directed graph',* ose digraf *'digraph'*, ka skaje të drejtuara (d.m.th., pikat *uv* të skajit nga nyja *u* në nyjen *v*, por nuk ka domosdoshmërisht një skaj *vu*). Një lak është një lidh që lidh një nyje të vetme me veten. Grafët gjithashtu mund të kenë skaje paralele, ose të shumëfishta midis dy nyjeve të njëjta (figura 16). Grafët e tillë quhen multigrafë *'multigraph'*, ose multidigrafë *'multidigraph'*, nëse ata janë të drejtuara. Një graf i padrejtuar është i lidhur nëse çdo nyje e tij mund të arrihet nga një nyje tjetër. Një graf i drejtuar është i lidhur fort nëse ai mund të arrihet nga secila prej nyjeve të tij në çdo nyje tjetër, dhe anasjelltas.

Një itenerar *'path'* është një sekuencë e renditur e skajeve që lidh disa sekuenca të renditura nyjesh. Dy itenerare janë të ndara brenda nyjeve nëse nuk kanë nyje të përbashkëta, përveç pikave përfundimtare. Skajet e një grafi me peshë *'weighted graph'* kanë një atribut për të përcaktuar disa vlera, të tilla si rëndësia ndërmjet nyjeve të lidhura. Distanca ndërmjet dy nyjeve është numri i skajeve në itenerarin ndërmjet tyre, ndërsa distanca 'me peshë' *'weighted distance'* është shuma e skajeve të itenerarit (figura 15).

Shkenca e rrjeteve është studimi i grafeve në jetën reale – kështu, rrjetet trashëgojnë terminologjinë e teorisë së grafeve. Rrjetet më të mëdha të botës reale janë komplekse dhe me interes më të veçantë janë rrjetet komplekse hapësinore – domethënë, rrjete komplekse me nyje dhe/ose skaje të pozicionuara në hapësirë (Gastner & Newman, 2006) (O'Sullivan, 2013). Një

rrjet rrugësh është një shembull i një rrjeti kompleks hapësinor me nyje dhe skaje të vendosura në hapësirë, siç janë hekurudhat, rrjetet e energjisë dhe rrjetet e ujit dhe kanalizimeve (Barthélemy, 2011).

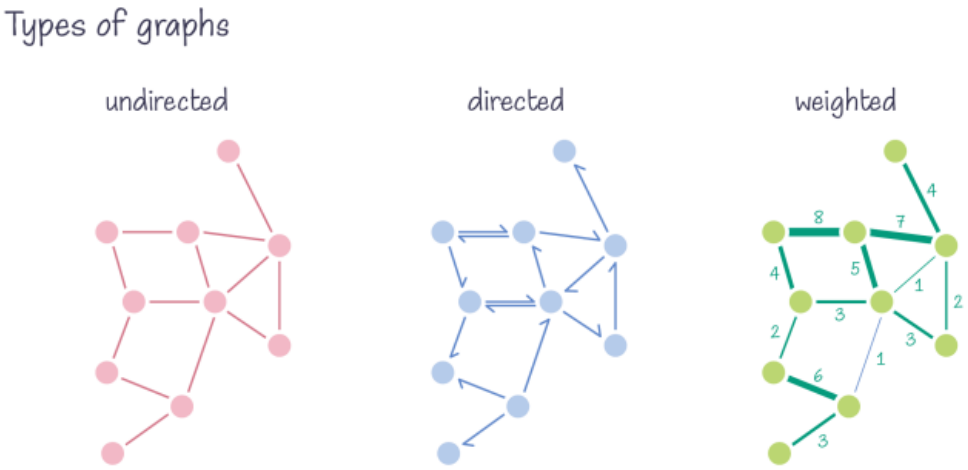


Figure 16 Tipet e grafeve në një rrjet, i padrejtuar, I drejtuar dhe me peshë. Burimi: web[24]

Një rrjet hapësinor është planar nëse mund të paraqitet në dy dimensione me skajet e tij duke u kryqëzuar vetëm në nyje (Strano, et al., 2013). Një rrjet rrugësh, për shembull, mund të jetë planar (veçanërisht në shkallë të vogla), por shumica e rrjeteve të rrugëve janë joplanare si p.sh. në autostrada, mbikalime, ura dhe tunele. Pavarësisht kësaj, shumica e studimeve në rrjetet rrugore urbane i paraqesin ato si graf planar (Strano, et al., 2013) për transportueshmëri sepse urat dhe tunelet janë vepra më të rralla, dhe kështu rrjetet janë afërsisht planare.

Rrjetet komplekse janë studiuar gjerësisht nga studiues dhe planifikues urban. Nga një këndvështrim cilësor, (Castells, 2009) argumenton se të kuptuarit e flukseve dhe rrjeteve, dhe jo vetëm vendndodhjet, janë kritike për të kuptuar qytetet. Nga një këndvështrim sasior, Batty (Batty M. , 2013 b) (Batty M. , 2013) e vendos modelimin urban në kontekstin e evolucionit të rrjetit.

Kur një rrjet transporti paraqitet konvencionalisht si një graf, lidhjet në rrjet bëhen skaje në graf, dhe kryqëzimet apo qytetet bëhen nyje të grafit (Kansky, 1963). Kështu është e mundur të përdorësh tregues të ndryshëm teorik të grafeve për të analizuar strukturën e rrjetit dhe për të cilësuar përbërës si lidhshmëria. Në përgjithësi, analizat teorike të grafeve mund të thuhet se përdorin nyje për të paraqitur elementët primar, dhe skajet për të paraqitur marrëdhëniet midis këtyre elementëve. Në tre rastet e para (figura 17), nyjet qartësisht paraqesin elementët primarë: tokat konvencionale, njerëzit dhe dhomat. Skajet paraqesin marrëdhëniet midis këtyre: ata janë efektivisht sekondare. Për shembull, marrëdhëniet e brendshme profesionale nuk ekzistojnë pa vetë profesionistët. Kjo qasje i përshtatet efektivisht situatës, kur ne dëshirojmë të fokusohemi të nyjet, dhe të dallojmë një hierarki të nyjeve. Në rastin e rrjeteve të transportit, elementet primar mund të jenë nyje (qytetet) të cilët bashkohen me linja të lëvizjes; ose elementet primare

[24] https://sitn.hms.harvard.edu/flash/2021/graph-theory-101/

mund të jenë linja të lëvizjes, të bashkuara në nyje (kryqëzime). Në çdo mënyrë, të dyja paraqiten nga një graf në të cilin nyjet janë kulme dhe linjat e lëvizjes janë skaje (figura 17),.

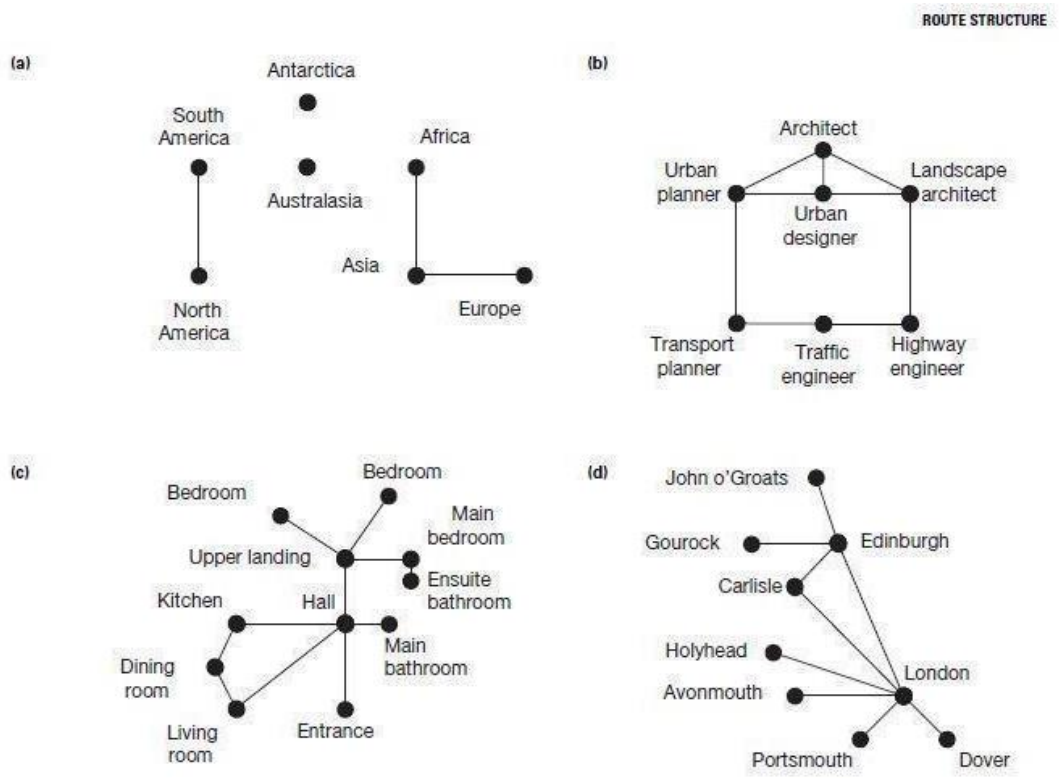


Figure 17 Paraqitjet e grafëve (a) tokat (b) Njerëzit (c) Dhomat (d) Rrjetet. Burimi: (Marshall S. , 2005)

Kjo formë përfaqësimi i përshtatet analizës së rrjeteve të linjave ajrore, rrjeteve hekurudhore/rrugore, ku nyjet janë pika të rëndësishme të qëllimit final ose të shkëmbimit. Ai gjithashtu u përshtatet itinerareve këmbësore në shkallë të vogla, ku itineraret janë formuluar në trajektore të ndryshme, nga pika në pikë. Në këtë mënyrë, nyjet janë pikat e fokusit dhe linjat e lëvizjes, të cilat mund të mos jenë pjesë të veçanta të infrastrukturës, janë të rëndësishme vetëm për aq sa ato përfaqësojnë marrëdhëniet midis nyjeve. Sidoqoftë, kjo marrëveshje është më pak efektive për rrjetet në të cilat itinerarët në vetvete janë fokusi kryesor. Për shembull, në rrjetet e rrugës, është shpesh statusi i tipave të ndryshme të itinerarit që kanë vëmendjen kryesore; kryqëzimet dhe shkëmbimet janë efektivisht nënprodukte të takimeve dhe kryqëzimeve të itinerareve (figura 18). Jo gjithmonë paraqitja e grafit teorik konvencional i cili bën dallimin midis llojeve të strukturave, është pikë interesi për planifikuesit urban dhe debateve për formën urbane.

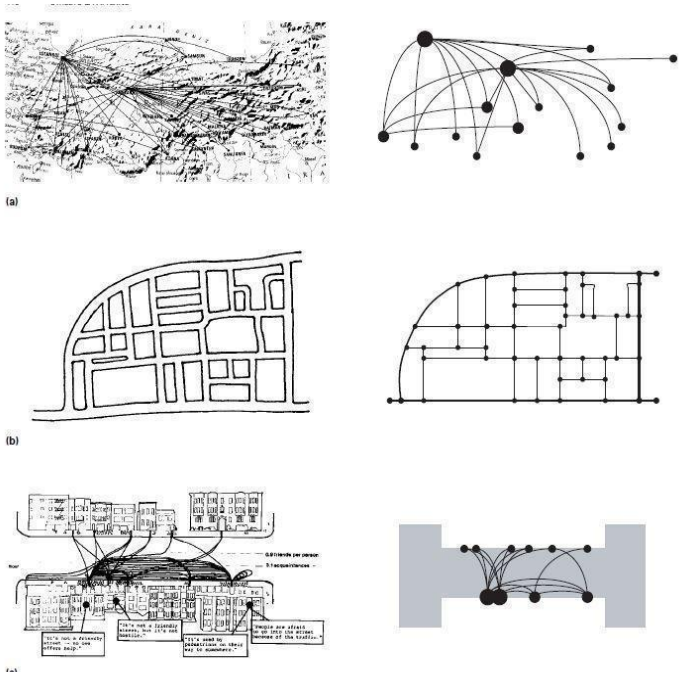


Figure 18 Rrjetet e qarkullimit në shkallë të ndryshme. Burimi: (Marshall S. , 2005)

Megjithëse paraqitjet (figura 19 (a) dhe (b)) mund të jepen për të analizuar lloje të ndryshme të shpërndarjeve në terma hapësinor urban, konfigurime e rrjetit të tyre që shfaqen si grafe që kanë të njëjtin numër të lidhjeve dhe nyjeve. Në këtë mënyrë, treguesit e grafeve teorikë jo gjithmonë veçojnë pjesë të veçanta të strukturës/hierarkisë që ato kanë, por megjithatë, dy paraqitje urbane të ndryshme mund të kenë të njëjtën strukturë të treguar në grafet teorikë (Figura 19). Kjo të sjellë nevojën e konsiderimit të formave alternative të analizave, të cilat tregojnë në mënyrë të saktë strukturën e rrjetit të rrugës urbane, duke vendosur linja të lëvizjes në qendër të skemës.

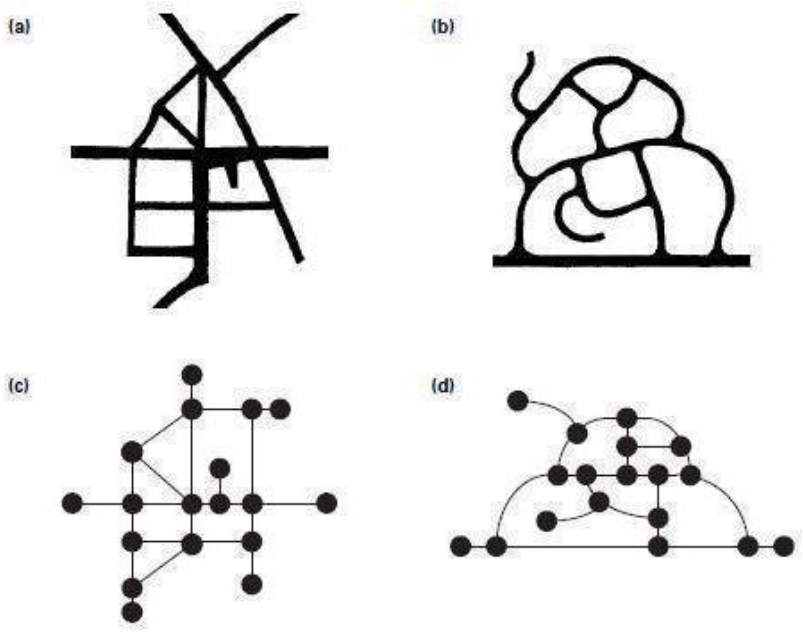


Figure 19 Dy rrjete rrugorë të krahasuar. Çdo rrjet ka 23 lidhje dhe 18 nyje. (a) 'rrjeti fokal tradicional. (b) "përdredhjet e shtrira" periferike. (c) Rrjeti I grafit (d) Rrjeti I grafit. Burimi: (Marshall S. , 2005)

2.5.5 Interpretimet kontekstuale të rrjeteve

Zgjedhja e përfshirjes së lidhjeve, dhe grumbullimi i atyre lidhjeve në itinerar, do të ndikojë se sa kuptimplotë janë rezultatet e analizës abstrakte të strukturës së itinerarit. Në analizën e strukturës së itinerarit, grumbullimi i lidhjeve brenda në itinerar supozohet të paraqes kalimet (rrugët) më të vazhdueshme të lëvizjes përmes një kryqëzimi. Një sugjerim për të përcaktuar modele të sakta të grumbullimit, do të sjell një rritje të domethënies së strukturave të itinerarit si më poshtë:

(i) Aty ku ekziston një klasifikim rruge i njohur dhe i vizatuar, ky klasifikim mund të përdoret për të përfituar itineraret. Kështu, në çdo kryqëzim (vendtakim), një rrugë e vetme mund të përzgjidhet nga dy lidhje me të njëjtin vizatim itinerari.

(ii) Aty ku struktura e itinerarit nuk është e zgjedhur nga (1) apo (2), atëherë vazhdimësia e grupimit/bashkimit/grumbullimit fizik mund të përdoret për të përzgjedhur itinerarin në vazhdim. Kjo taktik mund të jetë e dobishme kur je duke punuar (vetëm) nga një plan më shumë se sa nga një përvojë në shesh (site). Kuptime të tjera të mundshme për përcaktimin e itinerarit në vazhdim mund te jetë vazhdimi i emrit të rrugës, apo i modeleve të rrjedhës së trafikut, kur ato njihen.

Figura 20 demonstron palën më të hershme të shpërndarjes, kë të here është analizuar duke përdoru analizën e strukturës së itinerarit. Këtu, grafet themelore (e) dhe (f) janë ndryshe, cilado mënyrë (e) dhe (f) mund të deformohet, është e qartë që konfigurimet e tyre nuk janë të njëjta. Është ky ndryshim, që lind nga njohja e itinerareve të vazhduar, lejon strukturën e itinerarit të përzgjidhet, dhe si rrjedhojë të analizohet.

Në mënyrë efektive, problem kryesor është nëse seti (bashkimi) i konfigurimeve të grafeve këtu (Figura 20 (e) dhe (f)) siguron një paraqitje më të mirë të vendosjes së rrugëve së sa grafet e marr duke përdorur analizat konvencionale të rrjetit të transportit apo sintaksës së hapësirës.

Në këtë kuptim, ne mund të shënojmë se strukturat e grafit të paraqitjes së strukturës së itinerarit tregojnë qartësisht kontrastin midis Figurës 20 (e) e cila shpreh natyrën "fokale (kryesore)" të rrugës kryesore duke u lidhur me të gjithë rrugët e tjera; ndërkohë që Figura 20 (f) shpreh efektin izolues të shpërndarjeve të hierarkisë, më një *culs-de-sac*[25] në fund në disa fshirje nga rruga kryesore. Kështu, kjo jep një paraqitje të mire të kuptimit të mbajtur në mënyrë intuitive i tipareve të rëndësishme të modeleve të rrugës, jo vetëm lidhshmëria e tyre, por dhe vazhdimësia e itinerareve dhe hierarkia e thellësisë së tyre.

[25] Rrugë pa krye

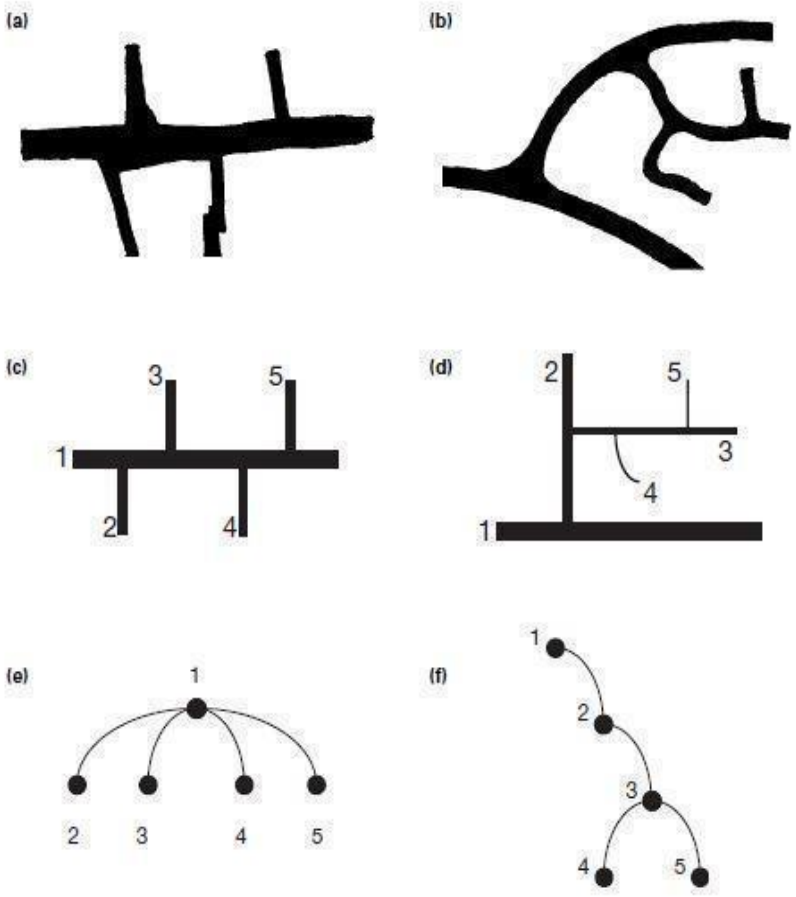


Figure 20 Dy shtrirje rrugësh të diferencuara nga analiza e strukturës së itinerarit. (a) Rrugë kryesore. (b) rrjet i degezuar. (c) Paraqitja e strukturës së itinerarit. (5 itinerare,4 bashkime). (d) Paraqitja e strukturës së itinerarit (5 itinerare, 4 bashkime). Burimi: (Marshall S. , Streets and Patterns, 2005)

2.6 Karakteristika të rrjeteve

Dendësia dhe lidhshmëria

Një rrjet rrugor konsiderohet si një model i mirë zhvillimi, nëse është tërheqës dhe i përshtatshëm për këmbësorët. Kjo arrihet duke siguruar një dendësi të imët dhe lidhshmëri të lartë. Sipas Congress for New Urbanism (2016), densiteti i kryqëzimeve në SHBA nis që nga 60 kryqëzime për mile^2, dhe shkon deri ne 500 në disa qytete të tjera. Blloqe më të vogla (ose më shumë kryqëzime për milje katrore) zakonisht janë më të rehatshme për këmbësorët, duke ofruar rrugë më të drejtpërdrejta drejt destinacioneve dhe artikuluar një mjedis në shkallë më njerëzore. Pra, ka një diskutim të patjetërsueshëm, që rrjeti duhet ti përgjigjet shkallës njerëzore. Një rrjet më afër shkallës njerëzore do të thotë më i mirë për këmbësorët, vend më i përshtatshëm për të ecur me biçikleta dhe më i sigurt për të ngarë makina.

Këto dy koncepte nuk mund të shihen të ndara nga njëri tjetri. Ka shumë matje që janë përdorur për të përshkruar veçoritë dalluese të rrjetit dhe lidhshmërinë e tyre Shumica e tyre fokusohen në llogaritjen e shkallës, dendësisë, numrit të segmenteve rrugore, gjatësive të blloqeve, dhe

nyjeve të përfshira në një rrjet. Këto parametra shërbejnë për të përshkruar formën urbane nën optikën e rrjeteve.

Teoria e grafeve na jep një zgjidhje të thjeshtë për unifikimin dhe vlerësimin e aspekteve komplekse të lidhshmërisë në pëlhurën urbane dhe mund të përdoret për të analizuar atë në nivel struktural dhe funksional (Urban & Keitt, 2001).

Qendërsia

Lidhur me konceptin e qendërsisë ka një numër të madh kërkimesh që e lidhin atë me fushën e planifikimit. Pyetja që ngrihet është se cilat janë nyjet që kanë qendërsi më të lartë në një rrjet? Koncepti i qendërsisë të ndërmjetëm *(betweenness centrality)* i përdorur fillimisht nga (Freeman, 1977), nënkupton çdo dy nyje në një rrjet shkëmbejnë mesazhe me probabilitet të barabartë në njësinë e kohës. Dhe ky shkëmbim ndjek gjithmonë rrugën më të shkurtër. Në thelb kjo nënkupton gjetjen e nyjes apo nyjeve më influente në rrjet, me anë të përllogaritjeve të rrugëve më të shkurtra që kalojnë në këtë nyje.

$$\text{betweennessy [i]} = \sum_{j,k \in G-\{i\}; d[j,k] \leq r} (\text{njk[i]}/\text{njk} * \text{W[j]})$$

ku n_{ik} është numri i rrugëve më të shkurtra nga nyja 'j' në nyjen 'k', në grafin 'G', dhe n_{ik}[i] është nëngrupi i këtyre rrugëve që kalojnë përmes 'i', me 'j' dhe 'k', i shtrirë në rrjetin me rreze r nga pika i, dhe W[j], është pesha e një destinacioni 'j'. Nyjet me qendërsi të ndërmjetëm më të lartë janë gjithashtu ato, heqja e të cilave nga rrjeti do të prishë më shumë komunikimet midis nyjet e tjera, sepse ato shtrihen në numrin më të madh të rrugëve që shkëmbehen më njëra tjetrën.

2.7 Kufizime në analizimin e rrjeteve

Literatura empirike e deritanishme, për shkak të kufizimeve në disponueshmëri, qëndrueshmëri dhe teknologji të të dhënave të rrjetit rrugor, vuan nga mangësitë e mëposhtme:

Së pari, madhësitë e kampionëve në studime priren të jenë mjaft të vogla për shkak të vështirësive të marrjes së të dhënave më të mëdha. Shumica e studimeve tentojnë të analizojnë deri në 50 kampionë për të studiuar transportin në shkallën e qytetit ose lagjes (Marshall & Garrick, 2010); (Strano, et al., 2013). Sigurimi i burimit të të dhënave nga institucione shtetërore mund të jetë i vështirë dhe kërkon kohë. Sidoqoftë, madhësitë e vogla të kampionëve mund të kufizojnë nivelin e përfaqësimit dhe besueshmërinë e rezultateve.

Së dyti, studimet zakonisht thjeshtojnë paraqitjen e rrjetit të rrugës në një graf planar (Barthélemy & Flammini, 2008); (Masucci, D, Crooks, & Batty, 2009). Në mënyrë tipike, studiuesit mbledhin rrjetet e rrugëve në një lloj objekti teorik grafesh nga të dhënat e GIS, për shembull duke ndarë linjat qendrore të të gjitha rrugëve në një zone studimi kudo që kalojnë në dy dimensione. Këto vija të ndara bëhen skaje dhe pikat e ndarjes bëhen nyje. Sidoqoftë, kjo metodë supozon një graf planar: urat dhe tunelet bëhen pika të ndarjes (dhe kështu nyje) edhe nëse rrugët nuk kryqëzohen në tre dimensione. Në qoftë se rrjeti i rrugëve nuk është në të

vërtetë i rrafshët, thjeshtimi planar prodhon një paraqitje më pak se idealja që mund të japin metrika të pasakta, të nënvlerësojë gjatësitë e skajeve dhe të mbivlerësojë numrin e nyjeve. Pra nëpërmjet këtij thjeshtimi, mund të modelohet në mënyrë të arsyeshme një rrjet rrugor në një qendër të qytetit mesjetar evropian, por modelon dobët rrjetin rrugor në qytetet moderne, të cilat kanë shumë autostrada, ura dhe tunele të ndara nga shkalla, në një rrjet jo-planar.

Problemi i tretë është përsëritshmëria. Duke qenë se në analiza të ndryshme merren vendime për kufizime të ndryshme të tilla si shtrirja hapësinore, thjeshtimi dhe korrigjimi topologjik, përkufizimet e nyjeve dhe skajeve, këto e bëjnë riprodhimin sfidues. Për shembull, studime të ndryshme ekzaminojnë qytetet deri në skajet urbane, por nuk shpjegojnë saktësisht se si dhe ku u përcaktua kjo periferi (Strano, et al., 2013). Disa studime nuk raportojnë nëse në rrjetet e tyre janë të përcaktuara drejtimet e lëvizjes së rrugëve ose jo (Porta, Vito, & Latora, 2006); (Strano, et al., 2013), mund të kenë pak rëndësi për studimet e këmbësorëve, por në interpretimin e transportit këto vlera mund të ndikojnë konsiderueshëm.

Studime të ndryshme marrin vendime të ndryshme, lidhur me përcaktimet e skajeve, kryqëzimeve, rrugëve pa krye, drejtimeve të lëvizjes etj (Marshall & Garrick, 2010); (Sevtsuk & Mekonnen, 2012) . Përkufizimet e tyre ndikojnë në mënyrën se si ne interpretojmë karakteristika të ndryshme të llogaritura si shkallë ose dendësia e kryqëzimit, dhe çdo vendim tjetër mund të krijojë probleme në përsëritjen, interpretimin dhe përgjithësimin e rezultateve.

Së katërti, siç u diskutua, moria e mjeteve dhe metodave nuk ofron një kornizë ideale që mund të ekuilibrojë madhësinë e kampionëve, përsëritshmërinë, përshtatshmërinë në ndërtimin dhe analizimin e të dhënave të rrjetit. Të marra së bashku, këto kufizime e bëjnë të vështirë punën e studiuesve të rrjetit rrugor dhe ë përfundimet që mund të nxirren nga këto studime.

2.8 Përfundime për zhvillimin e mëtejshëm të studimit

1. Tre përbërësit themelorë të kërkimit morfologjik urban janë forma urbane, hierarkia dhe koha

(i) Forma urbane përcaktohet nga tre elementë themelorë fizikë: a) ndërtesat dhe hapësirat e tyre të lidhura, b) parcela ose toka, dhe c) rrugët; (ii) Forma urbane mund të kuptohet në nivele të ndryshme të elementëve përbërës. Zakonisht, katër janë të njohura, që korrespondojnë me ndërtesën / parcelën, rrugën / bllokun, qytetin dhe rajonin; (iii) Forma urbane mund të kuptohet vetëm historikisht pasi elementet prej të cilave përbëhet, i nënshtrohen transformimit dhe zëvendësimit të vazhdueshëm.

2. Për shkak të kompleksitetit të lartë të fenomeneve urbane, analiza mund të thjeshtohet ta bazojmë në tre elementë kryesorë: (i) Rrjetet; (ii) Ndërtesat publike; (iii)Blloku urban

3. Qytetet janë një shembull thelbësor i sistemeve komplekse adaptive.

4. Për rrjedhojë, studimi i qyteteve si sisteme komplekse mund të jetë shumë sfidues për faktin se një qytet më kompleks edhe problemet e tij qe kërkojnë zgjidhje janë po aq komplekse. Vihen re tre tendenca të qarta që së bashku përcaktojnë fushën e studimeve të kompleksitetit

urban (Batty & Marshall, 2011): (i) Një qasje më sasiore, duke përdorur qasjet e modelimit stimulimet kompjuterike; (ii) Një qasje më cilësore që thekson filozofit e shkencës; (iii) Nocioni që përkufizon si përcaktimin, ashtu edhe optimizimin e një sistemi në një mënyrë reduktive, nga lart-poshtë, nuk është më 'një metaforë e përshtatshme për zgjidhjen e problemeve njerëzore', e cila gjithashtu prezanton paradoksin e dukshëm të 'planifikimit për kompleksitetin'.

5. Adaptimi është një pjesë vendimtare e sistemeve komplekse. Për ta zgjeruar edhe me shumë, kapaciteti adaptiv është ajo pikë ku qyteti duhet të përshtatet për të qëndruar koherent. Sistemet komplekse ku adaptimi është një fenomen shumë i rëndësishëm, zakonisht referohen si sistemet komplekse adaptive (Holland 1995).

6. Qytetet karakterizohen nga fenomeni i vetë-organizim (Portugali J. , 2000) (Salingaros N. , 2005). Disa forca të brendshme ndikojnë në strukturën ose rritjen e sistemit dhe përbërësit e tij rrisin koherencën e brendshme të strukturës së tij për shkak të atij vet.

7. Ripërtëritja adaptive dhe qëndrueshmëria janë tipare komplekse të sistemit adaptues që lidhen në mënyrë të drejtpërdrejtë me vetë-organizimin, *feedback*-un dhe jolinearitetitn (Holling, 1973).

8. Forma urbane mund të analizohet në disa aspekte apo kategori matëse, siç janë: kompleksiteti vizual, kohor, hapësinor, si edhe strukturor (fraktal, rrjetet), Geoffrey D. Boeing (2017).

9. Të rilidhësh njerëzit në qytet përgjatë një serie shkallësh të njëpasnjëshme, kërkon rizbulim të rregullit të humbur fraktal të qyteteve historike. Për të analizuar drejt hapësirat, format urbane duhet të shohim në një tjetër ekuilibër.

10. Ka një diskutim të patjetërsueshëm, që rrjeti duhet ti përgjigjet shkallës njerëzore. Një rrjet më afër shkallës njerëzore do të thotë më i mirë për këmbësorët, vend më i përshtatshëm për të ecur me biçikleta dhe më i siguritë për të ngarë makina.

11. Ndër të tjera, rrjetet karakterizohen nga densiteti, lidhshmëria dhe qendërsia.

3 TË DHËNAT DHE METODOLOGJIA

3. TË DHËNAT DHE METODOLOGJIA

3.1 Hyrje

Për sa i përket analizës së rrjeteve urbane ka një vend gjithmonë e më të dukshëm në shkencën e rrjeteve që kur Leonhard Euler prezantoi qasjen e tij të famshme mbi problematikën në '*Shtatë Urat e Königsberg në 1736*' (Devlin, 2000) dhe (Bonchev & Buck, 200). Mënyrat sesi studiuesit i kanë studiuar rrjetet në kohë është shumë e larmishme dhe janë detajuar në kapitullin e konsideratave teorike. Vlen të ndalim në studimet e përqendruara në formën urbane, si pjesë e këtij studimi (Southworth & Ben-Joseph, 1997), (Fecht, 2012), (Strano, et al., 2013). E parë në kohë dhe me mungesën e digjitalizimit, analizimi i rrjeteve ka pasur kufizime lidhur me bazën e të dhënave, kampione të vegjël, thjeshtim i konsiderueshëm i strukturës së rrjeteve dhe mungesë e mjeteve kërkimore të qëndrueshme për tu përdorur (Boeing G. , 2017).

Për zhvillimin e metodologjisë dhe adresimin e sfidave në këtë disertacion është përdorur soft-i *OSMnx* (*OpenStreetMap* dhe *NetworkX*). *OSMnx* është një bashkim i *OpenStreetMap* dhe *NetworkX*, dhe vjen si një paketë e koduar në *Python*, e cila është e hapur për përdoruesit. Gjithashtu, përveç *NetworkX*, moduleve të *Python* i shtohen edhe librari të tjera si *Matplotlib, Numpy, Pandas e Geopandas*. Ky soft është krijuar nga Goeffrey Boeing, drejtues i Laboratorit Urban Data, Universiteti Berkley, Kaliforni, Shtetet e Bashkuara të Amerikës (Boeing G. , 2017) (Boeing G. , 2017a). Qëllimi i këtij soft-i është mbledhja e të dhënave në mënyrë të thjeshtë, të qëndrueshme, dhe bazuar në perspektivat e teorisë se grafeve, transportit dhe projektimit urban.

OSMnx kontribuon në pesë drejtime kryesore për kërkuesit (Boeing G. , 2017): 1) shkarkimi automatik i të dhënave sipas kufijve administrativ të qyteteve, apo kërkesa të tjera të përdoruesit, si dhe gjurmët e ndërtesave brenda tyre; 2) shkarkimi i paracaktuar sipas nevojave dhe ndërtimi i automatizuar i të dhënave të rrjetit rrugor nga *OpenStreetMap*; 3) korrigjimi algoritmik i strukturës së rrjeteve; 4) aftësia për të ruajtur skedarët e rrjeteve si *shapefile, GraphML* ose *SVG*[26]; dhe 5) aftësia për të analizuar rrjetet, duke përfshirë llogaritjen, projektimin dhe vizualizimin e rrjeteve, dhe llogaritjen e përmasave metrike dhe strukturore. Baza e krijuar nëpërmjet këtij softi, përfshin matje të thjeshta që zakonisht ndodhin në studimet urbanistike dhe të transportit, si dhe matje më të avancuara të strukturës së rrjeteve.

Frymëzuar nga modeli i bashkëpunimit masiv të *Wikipedia*-s, projekti për *OpenStreetMap* filloi në 2004 dhe sot ka arritur në mbi dy milion përdorues. Cilësia e të dhënave të saj është përgjithësisht mjaft e lartë (Haklay, 2010); (Over, Schilling, & Neubauer, 2010). Edhe pse mbulimi i të dhënave ndryshon në të gjithë botën, është përgjithësisht i mirë kur krahasohet me vlerësimet përkatëse nga *CIA World Factbook*[27] (Maron, 2015). *OpenStreetMap* është një projekt i përbashkët që ofron një bazë hartografike falas dhe të përpunueshme për publikun, me kontributorë nga e gjithë bota, ku përditësohen të dhënat kohë pas kohe. Vitet e fundit, kjo

[26] Scalable Vector Graphics është një skedar vektorial që ruan shkallën e vizualizimit.

[27] Organizëm që ofron informacion bazë në shumë fusha, për rreth 266 entitete botërore.

platformë ka një rol kryesor si për hartëzimin ashtu edhe për marrjen e të dhënave hapësinore (Corcoran, Mooney, & Bertolotto, 2013).Të dhënat e *OpenStreetMap* përfaqësojnë një lloj të Informacionit Gjeografik Vullnetar, i cili krijohet nga përdoruesit dhe azhurnohet në mënyrë të vazhdueshme. Me zhvillimet teknologjike, Informacioni Gjeografik Vullnetar është një nga burimet më të rëndësishme dhe me rritjen më të shpejtë të të dhënave masive gjeo-hapësinore (Goodchild, 2007), (Boeing & Waddell, 2016).

Kur vjen diskutimi lidhur me '*të dhënat masive'*[28], studiuesit ndahen në dy grupe, ku grupi i parë, i cilëson si një "klishe" dhe janë një lloj i të dhënave që janë kuptimisht të ndryshme nga të dhënat tradicionale dhe ato në shkallë më të vogël (Kitchin, 2016). Këto grupe masive të dhënash përfaqësojnë modele në shkallë të madhe hapësinore dhe kohore dhe mund të kenë përfshirje të rëndësishme në çështjet e planifikimit dhe kërkimit urban (Zook, Graham, Shelton, & Gorman, 2010). Ndërsa, grupi tjetër e ka kritikuar glorifikimin e të dhënave masive, qytetin inteligjent dhe paradigmat e "shkencës urbane" që ato fuqizojnë (Kitchin, 2017).

Sipas Kitchin (2017)*"qasjet shkencore ndaj qyteteve janë konsideruar si mjaft naive dhe në perspektivë afatshkurtër, duke prodhuar shpjegime dhe modele tepër të thjeshtuara, dhe një kuptim të kufizuar dhe kufizues të mënyrës se si funksionojnë qytetet dhe si mund të trajtohen çështjet urbane."* Sidoqoftë, Informacioni Gjeografik Vullnetar si të dhënat nga *OpenStreetMap* ofrojnë një optikë të re me të cilën mund të shqyrtohen qytetet dhe sistemet njerëzore.

3.2 Dizajni i eksperimentit përmes softit OSMnx

Ky kapitull përshkruan mënyrën sesi do të realizohet eksperimenti, duke u bazuar në sfondin teorik të trajtuar në kapitujt e mëparshëm. *OSMnx* shkarkon rrjetet e rrugëve për çdo sipërfaqe të kërkuar sipas kufirit, ndërton rrjetin, korrigjon strukturën e tij dhe më pas llogarit parametrat e kërkuar. Në diagramin e mëposhtme është treguar procesi i funksionimit të *OSMnx* (figura 21). Hapat nëpër të cilat do të kalojë eksperimenti do të jenë si më poshtë:

Shkarkimi i kufijve administrativ dhe gjurmët e ndërtesave: Me *OSMnx,* cdokush mund të shkarkojë forma urbane mjedisi të ndërtuar nga OpenStreetMap në një rresht të vetëm të kodit *Python* dhe t'i projektojë ato në UTM[29] në një rresht më shumë të kodit.

Gjithashtu, lehtësisht mund të përftohen poligone për hapësira të ndryshme me interes studimi, të tilla si lagje, qytete, qarqe apo dhe shtete (figura 22).

Këto rezultate mund të ruhen në formë skedarësh në një *hard disk*. Në mënyrë të ngjashme, gjurmët e ndërtesave mund të merren kudo nga *OpenStreetMap* (Speranza, 2016). Fillimisht, nëpërmjet shkarkimit të rrjeteve, duke përdorur softin *OSMnx*, realizohet vizualizimi i rrjeteve për hapësirën që kërkojmë. Më pas nëpërmjet matjeve sasiore dhe strukturore, vlerësohet kompleksiteti i tyre.

[28] big data

[29] UTM - është akronimi i Universal Transverse Mercator, një sistem planar i rrjetit koordinativ i emërtuar për projektimin e hartës në të cilën bazohet. Sistemi UTM përbëhet nga 60 zona, secila prej 6◦ gjatësi gjeografike në gjerësi. Zonat numërohen nga 1 në 60, duke filluar nga gjatësia 180◦ dhe duke u rritur në lindje.

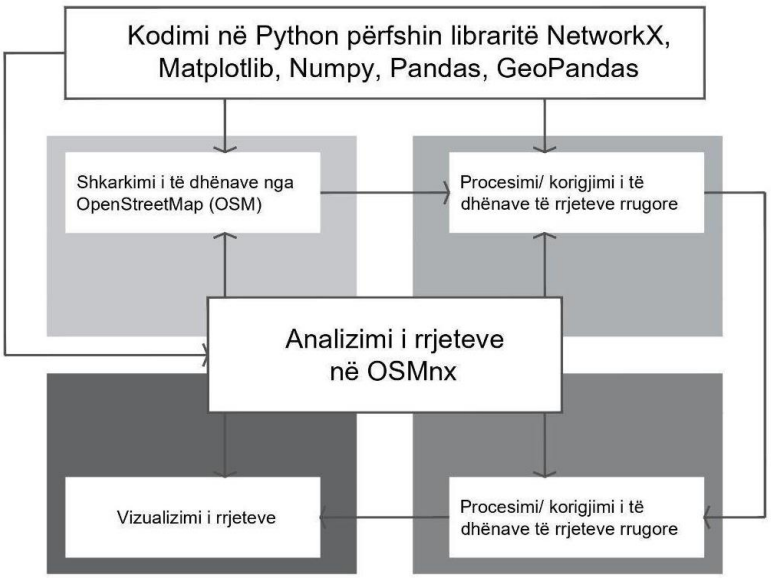


Figure 21 Diagrame mbi etapat që do të kalojë zhvillimi i eksperimentit. Burimi: autori

Figure 22 Kufijtë urbanë të qyteteve përkatësisht Tiranë, Korçë, Fier. Burimi: autori

Shkarkimi dhe ndërtimi i rrjeteve të rrugëve: Kontributi kryesor i *OSMnx* është shkarkimi dhe ndërtimi i rrjeteve të rrugëve. *OSMnx* lejon që të shkarkohen të dhënat e rrjetit rrugor dhe të ndërtojë strukturat e rrjeteve të korrigjuara, të projektojë e vizualizojë rrjetet dhe të ruajë rrjetin rrugor si *SVGs*, skedarë *GraphML* ose *shapefile* për përdorim të mëvonshëm. Rrjetet e rrugëve përfaqësohen si '*multidigraf*' dhe ruajnë drejtimin njëkahësh. Për më tepër, pasi të jetë shkarkuar rrjeti, *OSMnx* siguron funksione të integruara për të shkarkuar lartësinë e secilës nyje (*nga Google Maps Elevation API*) dhe për të llogaritur kuotat e rrugës.

Shkarkimi i rrjetit rrugor duke siguruar *OSMnx*, mund të realizohet në ndonjë nga mënyrat e mëposhtme (Boeing G. D., 2017):

- *një kuadrat kufizues*
- *një gjerësi gjeografike - plus një gjatësi/distancë në metra (ose një distancë përgjatë rrjetit ose një distancë në secilin drejtim kardinal nga pika)*
- *një adresë plus një distancë në metra (ose një distancë përgjatë rrjetit ose një distancë në secilin drejtim kardinal nga pika)*

• një poligon i kufijve të rrjetit të dëshiruar të rrugës
• një emër vendi ose listë e emrave të vendeve

Llojet e ndryshme të rrjetit mund të përcaktohen si më poshtë:

• rrugë automjetesh - të gjitha rrugët publike që mund të kalohet me mjete (por jo rrugët e shërbimit)
• rrugët e shërbimit - të gjitha rrugët publike që mund të kalojnë plus rrugët e shërbimit
• rrugë këmbësorësh - të gjitha rrugët që këmbësorët mund të përdorin (ky lloj rrjeti injoron drejtimin njëkahësh duke ndërtuar lidhje reciproke në të dy drejtimet ndërmjet secilës palë nyjesh të lidhura)
• rrugë biçikletash - të gjitha rrugët dhe shtigjet që çiklistët mund të përdorin
• të gjitha - të gjitha rrugët dhe shtigjet jo private të OpenStreetMap
• të gjitha private (all_private) - të gjitha rrugët dhe shtigjet e OpenStreetMap, përfshirë ato që janë private

Përvetësimi i rrjeteve të rrugëve nga emri i vendit ose nga poligoni është veçanërisht i dobishëm për studiuesit dhe planifikuesit. Kur i jepet si e dhënë një emër vendi, *OSMnx* gjeokodon emrin duke përdorur *API*[30] *Nominatim*[31] të *OpenStreetMap* dhe ndërton një poligon nga kufijtë e tij. Më pas e fut këtë poligon në një fashë prej 500 metra dhe shkarkon të dhënat e rrjetit rrugor brenda gjeometrisë së tij nga *OverStreetMap's Overpass API*. Gjithashtu, ndërton një rrjet rruge nga këto të dhëna, korrigjon strukturën, llogarit këndet e sakta dhe llojet e kryqëzimeve për çdo kryqëzim (kjo siguron që kryqëzimet të mos konsiderohen si rrugë pa krye, sepse mund të ndodhë që të lidhet me një nyje jashtë poligonit të dëshiruar), dhe më pas shkurton rrjetin në poligonin origjinal, të dëshiruar. Nëpërmjet këtij softi mund të kërkosh shumë lehtë një rrugë brenda një bashkie, qarku, shteti ose entiteti tjetër gjeografik. Dikush mund të kalojë një listë të vendeve (të tilla si disa qytete fqinje) për të krijuar një rrjet rrugor të unifikuar brenda bashkimit të gjeometrive të tyre. Këto operacione gjeohapësinore përfitojnë nga një indeks hapësinor i *'pemës R'* për të identifikuar shpejt nyjet që shtrihen brenda ose jashtë poligoneve (Guttman, 1984).

Korrigjimi dhe thjeshtimi i strukturës së rrjetit: *OSMnx* automatikisht korrigjon dhe thjeshton strukturën. Thjeshtimi është thelbësor për një analizë të saktë sepse nyjet në OpenStreetMap mund ta konfuzojnë atë në mënyrë kontradiktore: ato përfshijnë kryqëzime, por gjithashtu përfshijnë të gjitha pikat përgjatë një segmenti ku kurba e rrugës lakohet (figura 23). Këto të fundit nuk janë nyje në kuptimin e teorisë së grafeve, kështu që ne i heqim ato në mënyrë algoritmike dhe konsolidojmë skajet midis nyjeve të rrjetit (d.m.th., kryqëzimet dhe rrugët pa krye) në një skaj të vetëm të unifikuar (Boeing G. , 2017). Unifikimi i skajeve siguron që ato të ruajnë atributet dhe gjatësin si një segment i plotë i rrugës. *OSMnx* siguron modalitete të ndryshme të thjeshtimit për t'u siguruar studiuesve kontroll të imët për të përcaktuar nyjet në

[30] Në OSM, API shërben për marrjen dhe ruajtjen e të dhënave të papërpunuara në bazën e të dhënave OpenStreetMap
[31] nga latinishtja "me emër". Është një mjet për të kërkuar të dhënat e OpenStreetMap sipas adresës ose vendndodhjes.

mënyrë rigoroze. Pas thjeshtimit që pëson, një nyje është: 1. pika ku përfundon një rrugë pa krye; 2. pika ku përfundon një lak (rrugë e mbyllur); 3. kryqëzimi midis disa rrugëve ku të paktën një nga rrugët vazhdon përmes këtij kryqëzimi (d.m.th., nëse dy rrugë pa krye përfundojnë në të njëjtën pikë, duke krijuar një bërryl, pika nuk konsiderohet si një nyje)

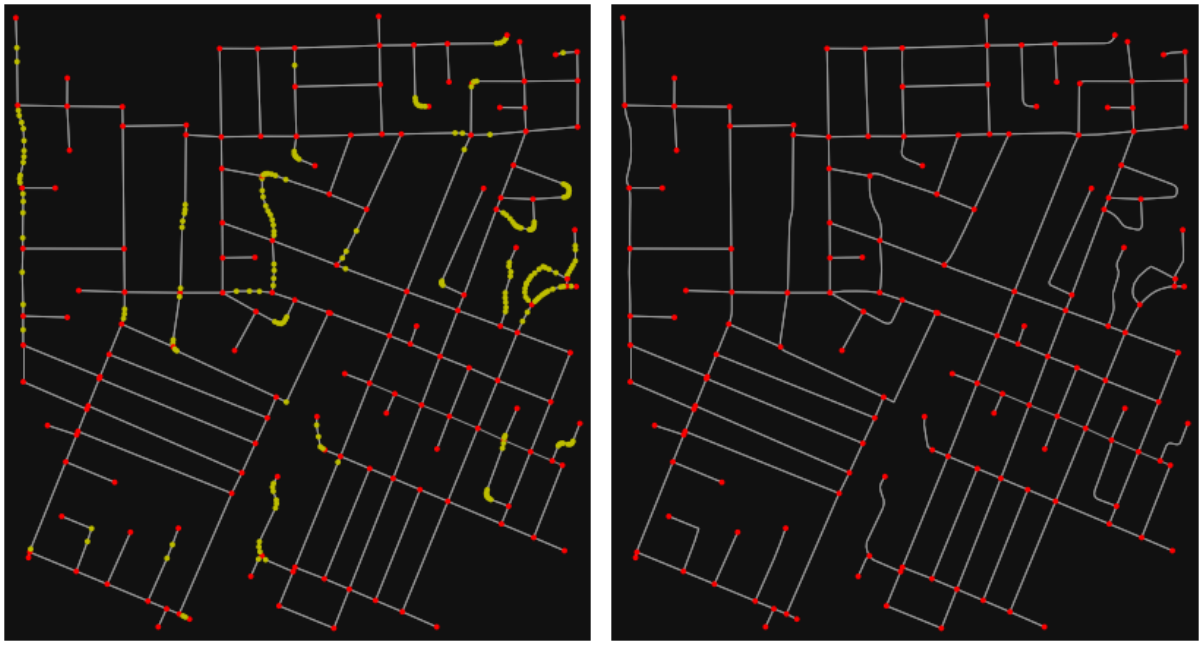


Figure 23 Rrjeti i një fragmenti të qytetit si një graf para thjeshtimit strukturor (majtas) dhe pas thjeshtimit (djathtas). Rrathet e kuq janë nyje dhe vijat e bardha janë skaje. Burimi: autori

Thjeshtimi ndodh, duke mbajtur vetëm ato nyje që përfaqësojnë rrugë pa krye, ose ku përfundon një lak i mbyllur dhe kryqëzimin e vërtetë midis disa rrugëve. *OSMnx* e bën këtë automatikisht në mënyrë rigoroze. Së pari, ai identifikon të gjitha nyjet që nuk janë kryqëzime dhe nuk janë rrugë pa krye. Pasi i heq ato, ruan me saktësi gjeometrinë hapësinore dhe atributet e segmentit të rrugës midis nyjeve të kryqëzimeve, sipas figurës së mëposhtme. Në mënyrë rigoroze, *OSMnx* konsideron se kryqëzimet me dydrejtime janë strukturalisht të njëjta me një rrugë të vetme që përkulet rreth një kurbe. Nëse për qëllime studimi, dëshirojmë ti mbajmë këto kryqëzime, mund të përdorim modalitet të tjera të ofruara nga programi.

Ruajtja e të dhënave të rrjeteve: *OSMnx* mund ti ruajë rrjetet si një skedar *GraphML*, për të punuar më vonë me to në softet e analizës së rrjetit si *Gephi* ose *NetworkX* (figura 24). Gjithashtu, nyjet dhe skajet ato mund të ruhen si skedarë *ESRI*, për të punuar më vonë në çdo soft standard GIS. Kur rrjeti ruhet si *shapefile*, grafi thjeshtësohet në një paraqitje pa përcaktuar senset e lëvizjes (e pa drejtuar). Sidoqoftë, drejtimi i nyjeve dhe origjina ruhen si atribute për përdorime të mëvonshme në GIS. Këto *shapefile* dhe skedarë *GraphML* mund të ngarkohen përsëri në *OSMnx*, për të shërbyer për përpunim, analizë ose vizualizim. *OSMnx* gjithashtu mund të ruajë rrjetet si skedarë *SVG* për përpunim të mëvonshëm në *Adobe Illustrator*.

Figure 24 Ruajtja e rrjetit rrugor për qytetin e Fierit, pas thjeshtimit të strukturës, ku dallohen qarte nyjet dhe skajet. Burimi: autori

Analizimi i rrjeteve: *OSMnx* analizon dhe llogarit statistikat e rrjeteve, përfshirë të dhënat metrike dhe strukturore në zonën e marrë në studim (tabela 2). Kështu përmes *OSMnx* llogariten parametrat si më poshtë: *n (numri i nyjeve), m (numri i skajeve), gjatësitë totale të rrugëve (m), gjatësitë totale të skajeve (m), numri i kryqëzimeve, mesatarja e gjatësive së rrugëve (m), mesatarja e gjatësive të skajeve (m), densiteti i nyjeve për km^2, densiteti i kryqëzimeve për km^2, densiteti i skajeve (km/km^2), densiteti i rrugëve (km/km^2), qarkullueshmëria mesatare, shkalla mesatare e nyjes, mesatarja e rrugëve për nyje, densiteti i nyjeve për km^2, densiteti i nyjeve të kryqëzuara për km^2, raporti i laqeve të mbyllura, qendërsia e nyjeve.* Këto parametra janë llogaritur në këtë disertacionin, lidhur me qëllimin e kërkimit.

Gjithashtu, nëpërmjet OSMnx mund të llogariten gjithashtu edhe: Shkalla mesatare e dy nyjeve fqinje, qëndërsia e nyjeve të ndërmjetme, qendërsia e mbyllur, shkalla e qendërsisë, koeficientët e grumbullimit, eksentriciteti, diametri, rrezja, shkalla e lidhshmërisë së nyjeve, lidhshmëria e skajeve, densiteti i nyjes për km^2, *PageRank*, proporcioni i skajeve me laqe të mbyllura, dendësia e rrugës për km^2, gjatësia totale e rrugës, gjatësia mesatare e rrugës, numri i segmenteve të rrugëve, numri mesatar i segmenteve të rrugëve që burojnë nga secili kryqëzim.., etj.

Numri i rrugëve për nyje dhe raportet që krijohen me to, nëpërmjet përdorimit të OSMnx përmirësohen krahasuar me studime të tjera. Shumica e studimeve thjeshtojnë shkallën e nyjeve

kur llogarisin sa skaje janë të mundshme në një nyje. Kjo ka disa probleme. *Së pari*, ato injorojnë faktin se një kryqëzim mund të ketë rrugë (skaje) të tjera të mundshme që lidhen me të edhe jashtë zonës së marrë në studim. Kështu, një kryqëzim me 4 rrugë afër periferisë mund të ketë vetëm tre prej tyre brenda zonës së marrë në studim, sepse i katërti lidh një nyje jashtë zonës dhe kështu cungohet studimi. *Së dyti*, laqet dykahëshe në një graf të drejtuar mund të llogariten si katër segmente fizike të rrugës të lidhura me kryqëzimin. Nëpërmjet këtij softi, krijon një algoritëm të ri për të numëruar saktë rrugët për nyje, duke marrë parasysh rrugët njëkahëshe dhe dykahëshe, skajet, laqet dhe kryqëzimet me rrugët që nuk shfaqen në grafik, sepse ato lidhen me një nyje jashtë kufijve.

Table 2 Tabelë mbi parametrat që mund të gjenerojë softi i përdorur. Burimi: autori

Të dhënat	**Interpretimi**
n (numri i nyjeve)	numri i nyjeve në një graf
m (numri i skajeve)	numri i skajeve në një graf
Gjatësitë totale të rrugëve *(m)*	Shuma e të gjitha gjatësive të rrugëve në sipërfaqen e zonës së marrë në studim, në metra
Gjatësitë totale të skajeve *(m)*	Shuma e të gjitha gjatësive të rrugëve (në një graf të drejtuar) në zonën e marrë në studim të mbuluar me rrjet
Numri i kryqëzimeve	numri i kryqëzimeve në sipërfaqen e zonës së marrë në studim të mbuluar me rrjet
Mesatarja e gjatësive së rrugëve (m)	gjatësia mesatare e rrugëve në sipërfaqen e zonës së marrë në studim të mbuluar me rrjet, në metra
Mesatarja e gjatësive të skajeve *(m)*	gjatësia mesatare e skajeve në graf, në metra
Densiteti i nyjeve për km^2	numri i nyjeve pjestuar me sipërfaqen e zonës së marrë në studim të mbuluar me rrjet
Densiteti i kryqëzimeve për km^2	numri i kryqëzimeve pjestuar me sipërfaqen e zonës së marrë në studim të mbuluar me rrjet
Densiteti i skajeve *(km/km²)*	shuma metrike e të gjithë gjatësive të skajeve të pjestuara me sipërfaqen e zonës në km^2
Densiteti i rrugëve *(km/km²)*	Shuma e gjatësive të rrugëve pjestuar me sipërfaqen në km^2
Qarkullueshmëria mesatare	Gjatësia totale e skajeve pjestuar me shumën e distancave më të mëdha ndërmjet nyjeve rastësore në çdo skaj
Shkalla mesatare e nyjes	mesatarja e numrit të skajeve rastësore në një nyje
Mesatarja e rrugëve për nyje	Mesatarja e numrit të rrugëve në një nyje
Raporti i laqeve të mbyllura	Raporti i laqeve të mbyllura (qarqe te mbyllura që kanë vetëm një skaj i cili fillon dhe mbaron tek vetja)

Qendërsia e nyjeve	Për çdo nyje, raporti i rrugëve më të shkurtra në kalojnë në një nyje

Vizualizimi i rrjeteve dhe formës urbane Në librin klasik të Allan Jacobs (1995) "Great Streets", mbi formën dhe projektimin urban në nivel rrugësh, janë paraqitur dhjetëra diagrama të vizatuara me dorë në stilin e hartave *Nolli*[32]. Secila prej tyre, përfaqëson një milje katror të rrjetit të rrugëve të një qyteti. Vizatimi i hartave të qyteteve në të njëjtën shkallë siguron objektivitet në hulumtimin dhe krahasimin vizual të rrjeteve dhe formave urbane të tyre. Ky vizualizim, si për rrjetet dhe për gjurmët e strukturave në qytete, mund të rikrijohet

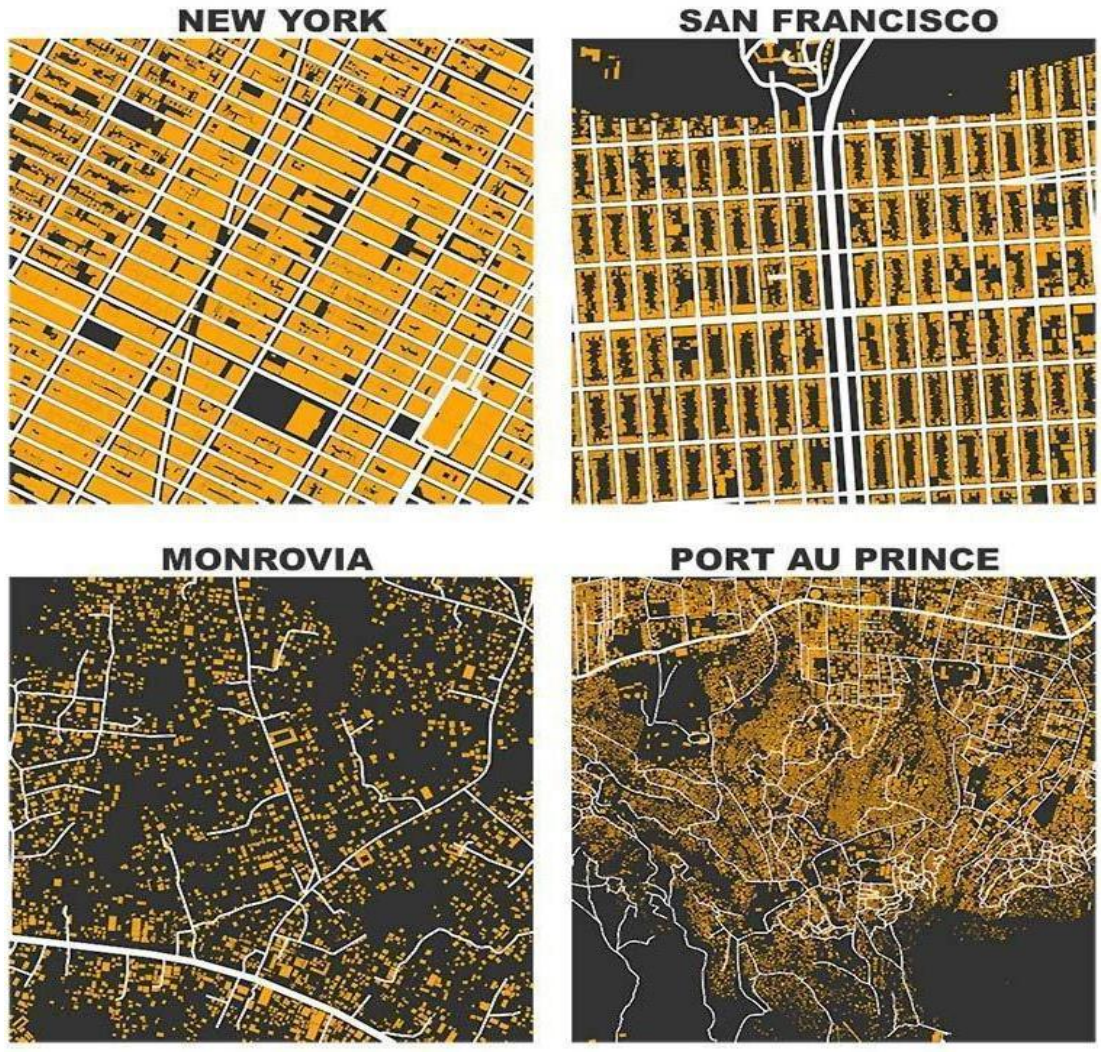


Figure 25 Module të gjeneruara nga OSMnx. Burimi: (Boeing G. D., 2017)

automatikisht nëpërmjet OSMnx.

OSMnx prodhon gjithashtu diagrama të rrjeteve të rrugëve dhe gjurmëve të ndërtesave ekzistuese si dhe analiza metrike për projektimin urban dhe planifikim nga studiuesit.

[32] Harta 'Nolli' është një vizatim plani dydimensional që përdoret për të kuptuar dhe dokumentuar aksesin dhe rrjedhën e hapësirës brenda një qyteti. Nolli i parë është vizatuar nga arkitekti italian Giovanni Battista Nolli, nga e ka marrë dhe emrin.

Vizualizimet në të njëjtën shkallë, na tregojnë se sa të ndryshme janë modelet e urbanizimit dhe paradigmat e tyre kur krahasohen me njëri tjetrin. Kjo mund të shërbejë si një mjet për të kuptuar rezultatet fizike të planifikimit dhe urbanizimit informal, por gjithashtu edhe për ti paraprirë procesit të planifikimit dhe projektimit urban në një mënyrë të qartë dhe të menjëhershme.

Këto përdorime mund të shihen ndoshta edhe më qartë kur përdorim *OSMnx* për të vizualizuar rrjetet së bashku me gjurmët e ndërtesave, siç tregohet në figurën më sipër. Dy imazhet e poshtme në figurën e mësipërme (figura 25), zbulojnë një mënyrë krejtësisht të ndryshme urbanizimi duke vizualizuar lagjet e varfra të Monrovia, Liberi dhe Port-au-Prince, Haiti (Boeing G. D., 2017). Këto vendbanime joformale kanë shkallë më të vogël, dhe nuk janë të strukturuara sipas logjikës së rrjeteve amerikane të rrugëve në dy imazhet e sipërme. Në zhvillimet joformale vihet re numër i madh objektesh për zonën e marrë në studim dhe gjurmë të vogla të strukturave. E kundërta ndodh në qytetet moderne të planifikuara.

Në përfundim, modelet e vizualizimit të ofruara nga *OSMnx*, të japin një model të saktë për të zhvilluar dhe studiuar analizat e morfologjisë urbane dhe rrjetet e qarkullimit që strukturojnë aktivitetet njerëzore dhe marrëdhëniet shoqërore.

3.3 Përzgjedhja e kampionëve

Duke u thelluar në konsideratat teorike të trajtuara në kapitullin e dytë dhe modelet teorike të leximit të formës urbane, përzgjedhja e kampionëve bëhet në funksion të tipologjisë së ndryshme të qyteteve dhe duke përfaqësuar rastet me tipike që karakterizojnë qendrat urbane shqiptare. Nëpërmjet vizualizimit të rrjeteve të realizuara nëpërmjet OSMnx, mund të bëhet një analizë e formës urbane të qyteteve në kompleksitetin e tyre. Për shkak të kufizimeve të softit të përdorur, këto të dhëna mund të gjenerohen vetëm brenda kufijve urbanë, ku dhe informacioni është më i madh. Më pas përmes një vlerësimi, përzgjidhen kampionë identifikues për secilin prej qyteteve, për të cilët duhet të sigurohemi që janë të krahasueshëm me njëri-tjetrin.

Karakteristikat e një rrjeti rrugor varen thelbësisht nga ajo çka qyteti shfaq, dhe në nivel hierarkik mund të jetë: kufiri administrativ i bashkisë, zona e urbanizuar (qyteti), në nivel lagjeje. Në rastin e parë, ka ndikim më të madh procesi vendimarrës nga lart-poshtë. Në rastin e dytë është një sistem i gjerë vetë organizues, i cili tenton të grumbullojë një heterogjenitet të formës së ndërtuar. Në rastin tjetër, shfaqet karakteri lokal i transformimit të formës urbane dhe ndërhyrjet nga poshtë-lart.

Ka një debat të gjerë lidhur me madhësinë e kampionëve që mund të analizohen, sepse madhësitë e tyre mund të çorientojnë rezultatet. Frankhauser dhe Batty (2008) pranojnë se shkalla apo madhësia e kampionit ndikon në vlerat e rezultateve të marra. Ata rekomandojnë se shkalla e qytetit mbetet një nga indikatorët kryesore që duhet të udhëheqë përzgjedhjen e kampionit. Duke qenë se analiza kryesisht fokusohet në vlerësimin e strukturës së rrjeteve dhe

pak më pak më masën e ndërtuar, panorama e përgjithshme e natyrës së qyteteve identifikohet në shkallë qyteti.

Shanken (2018) tregon sesi planifikuesit urban gjatë historisë kanë përdorur një mori metodash vizuale për të analizuar informacionin hapësinor dhe paraqitur qytetin. Kultura e të paraqiturit vizualisht të qytetit u ilustrua nga studimi i famshëm i Romës, i shekullit të XVIII-të të Giambattista Nollit, duke prodhuar hartat e famshme të Nolli-t, me përmasa 1 x 1 milje. Dy shekuj më vonë, Allan Jacobs (1995) në "Great Streets", përdori këtë metodologji për vizualizimin e dhjetëra rrugëve urbane në mbarë botën. Ky kampion dhe e njëjta metodologji vizualizimi, është përdorur nga Boeing (2020), për të përshkruar në mënyrë të ngjashme një milje katror të rrjeteve të rrugëve të shumë qyteteve, dhe për t'u përdorur më vonë për krahasim.

Sipas Salat (2011), ai ngre diskutimin se parametrat e formës urbane kanë kuptim krahasimor në nivel lokal dhe jo global. Qyteti nuk është një entitet që mund të trajtohet si një i vetëm, sepse ai nuk mund të shpërbëhet në pjesë që të mund të ruajnë karakteristikat e tij. Vetëm në metropolet moderne, mund të ketë vlerë ky qëndrim, pasi çdo kampion ruan njëtrajtshmërinë e të gjithit. Qyteti është shuma e një mori pjesësh të vogla të karakteristikat morfologjike të zonave apo lagjeve janë të ndryshme si në aspektin fizik ashtu dhe në atë social. Ky diferencim i hapësirave është vetë karakteri që qyteti ka. Në analizat e tij ai ka përcaktuar kampionë të rrjeteve 800 x 800 m, i cili largohet nga koncepti i qytetit si një i tërë. Sipas analizës së tij, qytetet mesdhetare përmbajnë shumë kampionë të tillë, të gjurmëve të qytetërimit romak. Ata të gjitha garantojnë uniformitet në leximin e formës urbane. Megjithatë ai thekson se nuk ka një rregull ose ligj të përcaktuar në madhësinë e kampionëve, uniformiteti i tij vjen nga rendi kompleks i karakterit fraktal të tij të krijuar nga fenomenet urbane. Faktaliteti morfologjik dhe hapësinor duhet të garantojnë uniformitetin e kampionit.

Në kontekstin shqiptar, në disertacionin e Denada Veizaj (2016), studimi ka sjellë një përmirësim të vlefshëm në nivel aplikativ, përcaktimin e madhësisë së kampionit urban. Është propozuar një metode që bazohet tek indeksi i uniformitetit të shpërndarjes se masës dhe observimit të vlerave të dimensionit fraktal. Nëpërmjet observimit të kurbës "scaling-behaviour", sipas Frankhauser-it, ka përcaktuar në mënyrë analitike zonat e thyerjes së homogjenitetit në një patern urban shqiptar. Zona e përshtatshme për të kryer matje teorikisht duhet të jete ajo zone ku uniformiteti është i lartë dhe vlerat e dimensionit fraktal mbeten relativisht të pandryshueshme. Në kampionët e analizuar në kontekstin shqiptar, në intervalin me brinjë]300, 500[të kampionit vërehet një rritje e vlerës së indeksit të uniformitetit, në të njëjtën kohe shihen lëvizje shumë të vogla të dimensionit fraktal. Kjo madhësi përkon edhe me identitetin kompozicional të paterneve urbane. Në rastin e indeve vernakulare përzgjedhja sipas madhësisë që përcaktojnë akset e lëvizjes rezulton e saktë.

Në këtë logjikë, pas analizimit të rrjetit të qyteteve sipas kufijve urbanë të tyre, përzgjedhja e kampionëve do të bëhet në madhësi 500 x 500 m, edhe sipas një këndvështrimi intuitiv të natyrës së këtyre qyteteve.

3.4 Metodologjia për matjet strukturale të formës urbane, rrjetet

Në këtë studim do të lexohet forma urbane, nën optikën e shkencave të rrjeteve. Rrjetet urbane do të konsiderohen si *multidigraph*, joplanare, të drejtuara. Të dhënat që do të gjenerohen do të analizohen për të vlerësuar kompleksitetin e rrjetit në aspektin e densitetit, lidhshmërisë dhe qendërsisë së nyjeve. Në ndihmë të këtyre, për të plotësuar më mirë spektrin e analizës, do të bëhet dhe një analizë e orientimit të rrugëve në qytetet shqiptare. Këto karakteristika të rrjetit janë atribute themelore në shkencat e planifikimit urban dhe ndikojnë në mënyrën se si forma fizike e një sistemi urban ndikon dhe strukturon ndërveprimet komplekse njerëzore dhe lidhjet, duke lidhur kështu strukturën dhe dinamikën e lëvizjes.

Studiues si Talen (2003), i kanë përdorur matjet e sipërcituara, për analiza të lidhura me morfologjinë urbane. Newman (2010) dhe Barthélemy (2011) kanë zhvilluar përkufizimet dhe algoritmet matematikore për këto matje, por megjithatë në këtë studim do bëhet një prezantim i shkurtër i tyre si koncepte, por jo një zhvillim i detajuar matematikor. Gjenerimi i rezultateve të tyre, bëhet nga softi i përdorur.

Matjet ndahen në matje metrike dhe strukturale, por vlen të përmenden në një graf joplanar, atributet metrike dhe strukturale janë ndërlidhura me njëra tjetrën (Masucci, Smith, Crooks, & Batty, 2009). Nuk pretendohet që zhvillimi i eksperimentit të ezauroj gjithë dimensionet e rrjeteve, por përpiqet të hedh dritë mbi një këndvështrim të caktuar, të lidhur me formën urbane dhe ciklin e qyteteve. Siç përshkruhet edhe në konsideratat teorike në kapitujt e mëparshëm, analiza do të zhvillohet në këto drejtime:

3.4.1 Densiteti i rrjeteve

Densiteti i referohet përqendrimit të elementeve për sipërfaqe, dhe në këtë studim konkretisht i referohet dendësisë së nyjeve, kryqëzimeve, skajeve dhe rrugëve. Matjet metrike të strukturës të rrjetit, janë ato që mund të maten me një pajisje të thjeshta të gjatësisë apo sipërfaqes dhe përfaqësojnë variablat e zakonshme në projektimin urban.

- **Mesatarja e gjatësive së rrugëve** *(average street length),* **dhe mesatarja e gjatësive të skajeve** *(average edge length),* shërben si një tregues për madhësinë e bllokut dhe tregon se sa i imët ose jo është rrjeti.

Nëse i referohemi Sevtsuk (2016), ai hap një diskutim tjetër lidhur me madhësinë e bllokut, duke eksploruar rrjetin si një atribut të formës urbane të lidhur me arritshmërinë e këmbësoreve. Ai studion se si parametrat e rrjeteve të rregullta ndikojnë në aksesin e këmbësorëve në destinacionet përreth, për një parcelë. Këta parametra përfshijnë gjatësinë e parcelës përgjatë rrugëve, thellësinë e parcelës, gjerësinë e rrugës dhe gjatësia mesatare të rrugës (gjatësinë e blloqeve) që përshkruhen nga numri i parcelave të vendosura në një bllok të vetëm. Ndërsa tre parametrat e parë kanë efekte të parashikueshme në arritshmërinë, sa më të vegjël janë zakonisht më mirë, parametri i fundit jo. Duke përdorur simulimet, ai tregon se marrëdhënia midis arritshmërisë së këmbësorëve dhe gjatësisë së bllokut është jolineare dhe parabolike. Kur kombinohen dimensione të ndryshme për sipërfaqen e parcelës dhe thellësinë e saj, dhe gjatësinë e bllokut dhe gjerësinë e rrugës, atëherë blloqet tipike më të vogla kanë tendencë të

gjenerojnë akses më të lartë për këmbësorët sesa blloqet më të mëdha. Në studimin tonë, ku në konsideratë janë marrë të gjitha rrjetet, ku ka lëvizshmëri në një qytet, nga matjet empirike të kryera përmes soft-it, sa të vogla të jenë gjatësitë mesatare të rrugëve aq më i imët do të konsiderohet rrjeti. Leximi i gjatësisë mesatare të rrugëve, lidhet me madhësinë e bllokut dhe formën urbane. Nëse i referohemi Jacobs (1995), madhësia e bllokut në qytete të ndryshme në botë, jepet sipas tabelës më poshtë:

Table 3 Madhësia mesatare e bllokut në qytete të ndryshme. Burimi: (Jacobs A. B., 1995)

Qyteti	**Madhësia mesatare e bllokut (m)**
Rome	58
Bologna (qendër)	68
Paris (Louvre)	75
Pompeii	68
New York (Manhattan)	84
NewYork (Midtown)	129
San Francisco (qendër)	108
San Francisco (Midtoën)	125
Los Angeles (qendër)	119
Irvine, CA	393

Ajo lidh qartazi, madhësinë mesatare të bllokut me karakterin e këtyre qyteteve.

- **Densiteti i rrugëve** *(street density)* është shuma lineare e të gjitha skajeve në paraqitjet e grafit të pjestuara me sipërfaqjen e zonës.
- **Densiteti i skajeve** *(edge density)* është shuma metrike e të gjithë gjatësive të skajeve të pjestuara me sipërfaqjen e zonës
- **Densiteti e nyjeve** *(the node density)* është numri i nyjeve pjestuar me sipërfaqen e zonës së marrë në studim të mbuluar me rrjet
- **Densiteti i kryqëzimeve** *(the intersection density)* është densiteti e nyjeve të cilat kanë më shumë se sa një rrugë që buron prej tyre (duke përjashtuar kështu rrugët pa rrugëdalje)

Këto katër parametra të densitetit ofrojnë tregues se sa i imët është rrjeti.

3.4.2 Lidhshmëria

Pjesë e matjeve strukturore të lidhshmërisë janë qarkullueshmëria mesatare, shkalla mesatare e nyjes, mesatarja e rrugëve për nyje, densiteti i nyjeve për km^2, densiteti i kryqëzimeve për km^2 dhe raporti i laqeve të mbyllura. Matjet metrike të lidhshmërisë të rrjetit varen nga mënyra se si paraqiten zonat e studimit dhe ato mund të kenë sjellje kontradiktore (Knight & Marshall,

2015). Në studimin e realizuar prej tyre, ata matin lidhshmërinë, densitetin e kryqëzimeve dhe densitetin e rrugëve, në zona të ndryshme studimore dhe me fizionomi gjeometrike të ndryshme. Duke kontrolluar shumë variabla, si madhësitë e blloqeve dhe gjeometrinë e tyre, rrugët e drejta, përmasat dhe gjeometrinë e rrjetit, ata arrijnë në përfundimin se sjelljet e këtyre metrikëve ndryshojnë ndjeshëm nga ato të synuara. Ato janë funksione jolineare si të zonës së studimit ashtu edhe të gjeometrisë dhe në fund të fundit janë kontradiktore dhe të paparashikueshme. Megjithatë, duke matur disa parametra të lidhshmërisë mund të krijohet një spektër më i gjerë i leximit.

- **Qarkullueshmëria mesatare** *(the average circuity)*, e cila pjeston shumën e të gjitha gjatësive të skajeve me shumën e distancave më të mëdha ndërmjet nyjeve rastësore në cdo skaj (Levinson & El-Geneidy, 2009), (Barthélemy, 2011), (Strano, Nicosia, Latora, Porta, & Barthelemy, 2012), (Giacomin & Levinson, 2015). Ky është raporti mesatar midis gjatësisë së një skaji dhe distancës së vijës së drejtë midis dy nyjeve që ajo lidh.

Shumica e studimeve në rrjetet rrugore, i kanë përafruar vlerat e qarkullueshmërisë mesatare rreth 1.2 (Newell, 1980) . Kjo shifër ndryshon nga një vend në një tjetër, për shkak të karakteristikave në densitetin e rrjetit. Një rrjet i lidhur në mënyrë të përkryer do i ketë vlerat e qarkullueshmërisë mesatare rreth 1.
Në figurën 26, jepet një krahasim midis një rrjeti dhe rrugës më të shkurtër sipas gjeometrisë

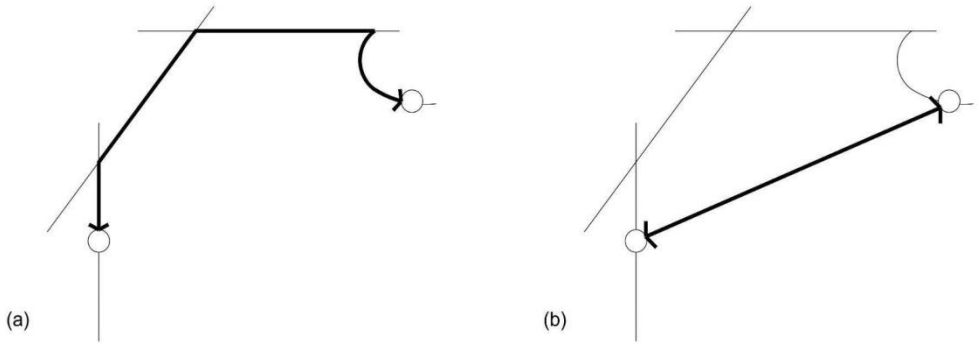


Figure 26 Krahasim midis a) rrjetit dhe b) rrugës më të shkurtër sipas gjeometrisë Euklidiane. Burimi: autori

Euklidiane, për të arritur nga pika e origjinës O, në destinacion pika D. Qarkullueshmëria është raporti midis gjatësisë së përshkruar të rrugës me atë të rrugës më të shkurtër sipas gjeometrisë Euklidiane, dhe jepet sipas raportit:

$C_{ij}=D^{n}_{ij} / D^{e}_{ij}$,

ku C_{ij} është qarkullueshmëria për të arritur nga origjina i në destinacionin j, D^{n}_{ij} është distanca minimale e kalimit në një rrjet, sipas figurës 1a) dhe D^{e}_{ij} është distanca euklidiane, vija e drejtë ndërmjet origjinës dhe destinacionit (Levinson & El-Geneidy, 2009). Kështu vlera minimale teorike e qarkullueshmërisë mesatare është 1, kur distanca e rrugës barazohet me atë euklidiane. Kur kjo vlerë është afër 1, do të thotë që efiçensa e qarkullueshmërisë është shumë e lartë. Duke qënë se pikat e origjinës dhe destinacionit mund të jenë rastësore, të dhënat e tyre kthehen si pika të një rrjeti. Në mënyrë të përmbledhur, nëse origjina dhe destinacioni i një rruge bien përkatësisht në rrjetin *i* dhe *j*, rrjeti *i* dhe *j* përdoren si udhëtim O_i dhe D_j, dhe ndërtohet matrica

OD për kryerjen e analizës. Atëherë qarkullimi mesatar i të gjitha pikave O të të dhënave GPS që udhëtojnë nga rrjeti O_i në rrjetin D_j, konsiderohet si qarkullueshmëria e rrjetit Oi.

$$C_i = 1/N_i \sum_{n=0}^{N} \quad Cn$$

Ku C_i është qarkullueshmëria në rrjetin O_i,; N_k është numri i të gjitha pikave të origjinës në rrjetin Oi; Cn është qarkullueshmëria e n pikave të origjinës O; C_i është qarkullueshmëria më e vogël e rrjetit O_i;

Në qoftë se qarkullueshmëria ndryshon, kjo do të thotë që ka ndryshuar shtrirja e rrjetit, përdorimi i tij lidhur me drejtimet, ose të dyja. Nëse vlera e qarkullueshmërisë është më e lartë do të thotë që rrjeti është më pak efiçent për sa i përket pikës më të shkurtër. Matja e qarkullueshmërisë mesatare, në studimin e kampionëve urban, lidhet me kohën e lëvizjes për të arritur nga një pikë në një tjetër, dhe mund të përdoret edhe për qëllime turizmi për të arritur në destinacion.

- **Shkalla mesatare e nyjes** *(the average node degree)*, ose numri mesatar i skajeve që ndodhin në secilën nyje, përcakton se sa mirë janë të lidhura nyjet. Sa më e lartë të jetë shkalla mesatare e nyjes aq më i lidhur është rrjeti.

Raportet e lidhura me to karakterizojnë llojin, dominimin dhe shpërndarjen hapësinore të lidhshmërisë të kryqëzimeve. Në mënyrë të ngjashme, lidhshmëria e nyjeve, lidhshmëria e skajeve, dhe cka është më e dobishme lidhshmëria mesatare e nyjeve, d.m.th., numri mesatar i nyjeve që duhet të hiqet për të shkëputur një çift nyjesh jo ngjitur të zgjedhur rastësisht, mund karakterizojnë se sa tërësisht i lidhur dhe i përshkueshëm është një rrjet. Matjet strukturale të rrjetit rrugor mund të tregojnë më fort lidhjen dhe qëndrueshmërinë e rrjetit dhe mënyrën e shpërndarjes së këtyre vlerave.

Indeksi β mat nivelin e lidhjes në një graf dhe shprehet me lidhjen ndërmjet numrit të lidhjeve (e) mbi numrin e nyjeve (v). β=e/v (Kansky, 1963). Pemët dhe rrjetet e thjeshta kanë vlerë β më të vogël se një. Një rrjet i lidhur në një cikël ka vlerën 1. Rrjetet më komplekse kanë vlerë më të madhe se 1. Në një rrjet me numër fiks nyjesh, sa më i madh të jetë numri i lidhjeve, aq më i lartë është numri i shtigjeve të mundshme në rrjet. Rrjetet komplekse kanë një vlerë të lartë të β-s (figura 27).

Në mënyrë të ngjashme, por më konkretisht, **rrugët mesatare për nyje** *(the average street per node)* matin numrin mesatar të rrugëve (d.m.th., skajet në paraqitjen e drejtuar të grafit) që burojnë nga secila nyje (d.m.th., kryqëzimet dhe rrugët pa krye). Kjo përshtat shkallën mesatare të nyjes lidhur me formën fizike më tepër sesa me qarkullimin apo rrjedhën. Shpërndarja dhe proporcioni i numrit të skajeve për nyje karakterizon llojin, prevalencën dhe shpërndarjen hapësinore të ndërlidhjes së kryqëzimit dhe rrugët pa krye në rrjet.

Lidhshmëria mat numrin minimal të nyjeve ose skajeve që duhet të hiqen nga një rrjet i lidhur për ta shkëputur atë (Urban & Keitt, 2001); (Dill, 2004)

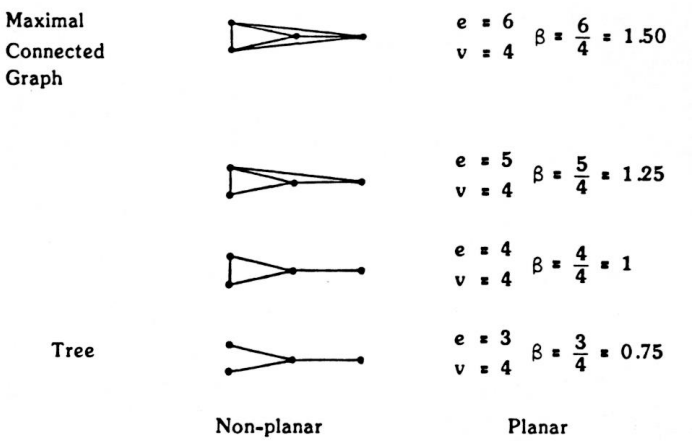


Figure 27 Diagramat e Kansky-t. Burimi: (Kansky, 1963)

Nëpërmjet lidhshmërisë arrihet matja e qëndrueshmërisë, pasi rrjetet komplekse me lidhshmëri të lartë ofrojnë më shumë alternativa kalueshmërie dhe rrjedhimisht mundësitë e ngërçeve janë më të pakta. Sidoqoftë, lidhja e nyjeve dhe skajeve është më pak e dobishme për rrjetet pothuajse planare si p.sh. rrjetet e rrugëve ku shumica e rrjeteve të rrugëve ka një vlerë lidhshmërie 1, pasi prania e një rruge të vetme pa krye tregon se heqja të paktën një nyjeje ose skaji do të shkëputë rrjetin. Përkundrazi, lidhshmëria mesatare e nyjeve të një rrjeti, nënkupton numrin mesatar të rrugëve të brendshme të nyjeve të shkëputura, ndërmjet çdo çift nyjesh, i cili paraqet numrin e pritur të nyjeve që duhet të hiqen për të shkëputur nyjet e njëpasnjëshme (Beineke, Oellermann, & Pippert, 2002), (Dankelmann & Oellermann, 2003). Raporti i laqeve të mbyllura është raporti i një skaji që lidhet vetëm në një nyje, qark i mbyllur që ka vetëm një skaj i cili fillon dhe mbaron tek vetja

Sa më i lartë të jetë ky raport, aq më shumë lidhshmëri paraqet rrjeti, është karakteristikë e indeve vernakulare.

- Matje të tjera të lidhshmërisë, të tilla si densiteti i nyjeve për km^2 dhe densiteti i kryqëzimeve për km^2, mund të krijojnë një kornizë të lidhshmërisë dhe qëndrueshmërisë së rrjeteve rrugore, më të mirë sesa lidhshmëria standarde e nyjeve dhe/ose skajeve.

Gjithashtu, mesatarja e gjatësive të rrugëve, është tregues i lidhshmërisë. Sa e shkurtër të jetë kjo gjatësi, aq më i lidhur paraqitet rrjeti.

E parë në këndvështrimin e ripërtëritjes adaptive që pësojnë qytetet, lidhshmëria është një atribut i saj, dhe përcakton që numrin minimal të nyjeve dhe skajeve që mund të hiqen nga një rrjeti që rrjeti të shkëputet. Ripërtëritja adaptive, është shpejtësia e një sistemi për të rikuperuar pas një ngacmimi. Rrjetet me lidhshmëri të ulët mund të kenë disa pika të dobëta, duke e lënë sistemin veçanërisht të brishtë. Kjo mund konstatohet në modelin urban përmes përshkueshmërisë dhe ngërçeve të mundshme: nëse qarkullimi orientohet drejt një pike e cila ka predispozitë të krijojë ngërç, për rrjedhojë do të kemi të bëjmë me bllokim, trafik dhe krijimin e një skeme qarkullimi që nuk funksionon.

3.4.3 Qendërsia

Qëndërsia i nyjeve *(betweenness centrality)* vlerëson rëndësinë e një nyje, me anë të përllogaritjeve të segmenteve më të shkurtra që kalojnë në këtë nyje (Freeman, 1977). Pra, matjet e qendërsisë tregojnë nyjet më të rëndësishme në një rrjet.

Në analizën e qendërsisë së nyjeve në nivel qyteti, shtrirja e tyre e shpërndarë sinjalizon një heterogjenitet të rrjetit, me ekzistencën e disa rrugëve shumë qendrore të cilat me shumë mundësi kanë probleme me fluksin dhe ngarkesën në rrjet. Kështu nyja që ka kontroll më të lartë në lëvizshmërinë ndërmjet nyjeve dhe grupeve të nyjeve ka qendërsi më të lartë. Nga ana tjetër, përqendrimi i zonave që kanë qendërsi të lartë, është në vetvete interesante pasi tregon për zona potencialisht të ngarkuara me flukse (figura 28).

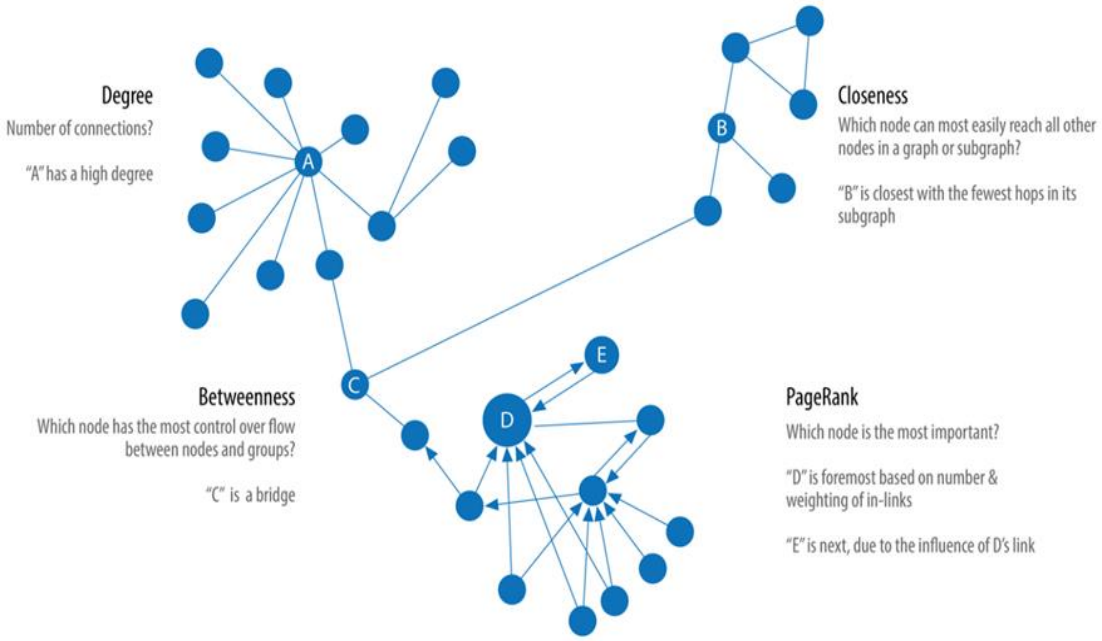


Figure 28 Diagrama mbi centralitetin e rrjetit. Burimi: https://neo4j.com/category/webinar/feed/

Për të shkuar më tej, zonat qendrore nga këndvështrimi gjeografik gjithashtu kanë probabilitet për të pasur qendërsi më të lartë të nyjeve. Megjithatë, nëse në një paraqitje të qytetit, shohim se disa rrugë apo zona mund të kenë qendërsi të lartë, tregon një modul kompleks të shpërndarjes së fluksit në qytet. Mesatarja e qendërsisë së nyjeve është mesatarja e gjitha qendërsive të nyjeve prezente në rrjetin e marrë në studim (Barthélemy, 2011).

Në veçanti, përqendrimi maksimal i qendërsisë së nyjeve në një rrjet specifikon përqindjen e rrugëve më të shkurtra që kalojnë përmes nyjes më të rëndësishme. Ky është një tregues i ripërtëritjes adaptive: rrjetet me një mesatare më të lartë të qendrave të ndërmjetme janë më të predispozuara të kenë ngërçe.

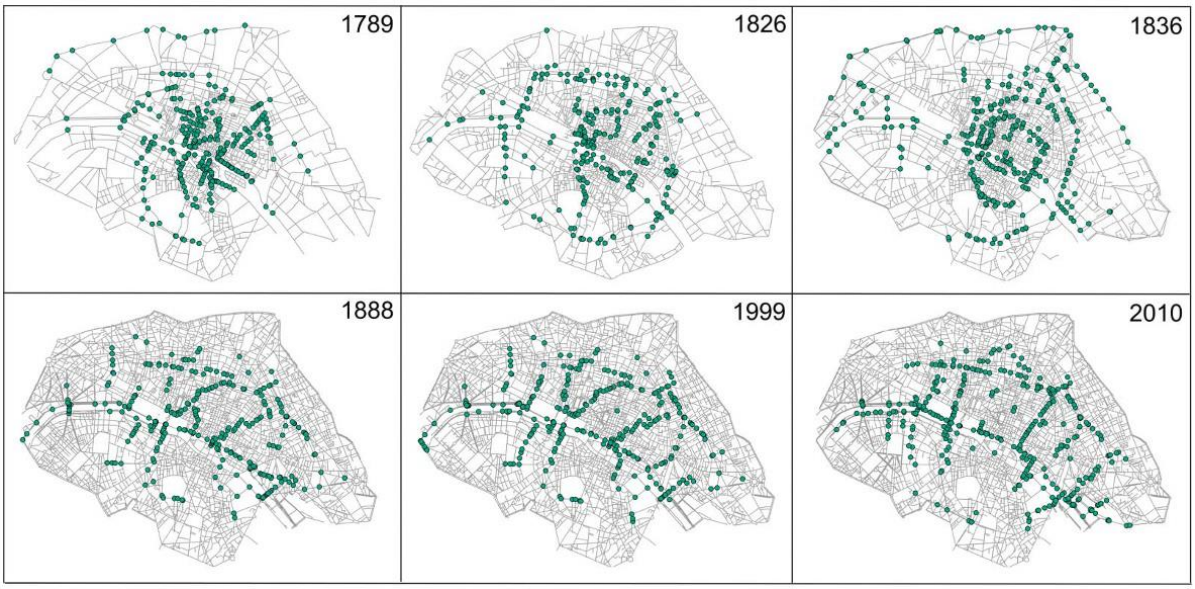
Barthelemy (2013) përdor analizat e qendërsisë për të identifikuar ndërhyrjet nga lart-poshtë kundrejt ndërhyrjeve poshtë-lart të vetë organizimit dhe evoluimit të strukturës urbane në Paris. Në studimet e tij, ai shqyrton rastin e evolucionit të rrjetit të rrugëve të Parisit gjatë më shumë

se 200 viteve me fokus të veçantë në shekullin e XIX-të, periudhë kur Parisi përjetoi transformime të mëdha nga Baron Haussmann. Në thelb, deri në mesin e shekullit të XIX-të, Parisi qendror kishte një strukturë mesjetare të përbërë nga shumë rrugë të vogla dhe mbipopullim. Në 1852, Napoleoni III ngarkoi Haussmann për të modernizuar Parisin duke ndërtuar rrugë më të sigurta, më të mëdha të lidhura me stacionet e reja të trenit, sheshe qendrore e simbolike, duke përmirësuar fluksin e trafikut dhe e fundit, por jo më pak e rëndësishme, sigurimin e qarkullimit të trupave të ushtrisë. Rasti i Parisit nën Haussmann ofron një shembull interesant ku ndryshimet për shkak të planifikimit qendror janë shumë transformuese të qytetit.

Qendërsia e ndërmjetme e një nyje në thelb mat fraksionin e kohës që përdoret një nyje e caktuar në rrugët më të shkurtra që lidhin çdo çift nyjesh në rrjet dhe është kështu tregues i kontributit të lidhshmërisë në organizimin e një rrjeti. Në rastin kur analizohet një pjesë e kufizuar e një rrjeti hapësinor, duhet të jenë dy efekte të rëndësishme për tu konsideruar:

Së pari, ndërsa marrim parasysh një pjesë, vetëm rrugët brenda kësaj pjese merren parasysh në llogaritjen e qendërsisë së nyjeve dhe kjo zakonisht nuk pasqyron realitetin. Si rezultat, qendërsi e ndërmjetme e nyjeve do të jetë në gjendje të zbulojë rrugë dhe nyje të rëndësishme brenda struktura e brendshme e rrjetit por do të mungojë në shkallë të gjerë, si rrugë komunikimi që lidh qytetin me pjesën e rrethinave.

Figure 29 Shpërndarja hapësinore e nyjeve me qendërsi më të lartë në Paris. Burimi: (Barthelemy, Viana, Strano, & Bordin, 2013)



Pika e dytë ka të bëjë shpërndarjen hapësinore të qendërsisë. Nyjet më qendrore janë afër bariqendrës së nyjeve. Qendërsia hapësinore dhe qendërsia e ndërmjetme janë zakonisht të ndërlidhura me njëra tjetrën. Kështu që, pozicioni i nyjeve me qendërsi më të lartë, ofron një përshkrim interpretues mbi zonat ku fluksi është më i madh.

Rezultatet për qendrën e Parisit tregojnë se shumica e treguesve ndjekin një ecuri të qetë evolucioni, i dominuar nga një proces densifikimi, pavarësisht nga “trazimi”që ndodhi gjatë

Haussmann. Në rezultatet e studimeve të tij, gjurma më e rëndësishme sasiore e planifikimit qendror është riorganizimi hapësinor i nyjeve më qendrore (figura 29).

Ai tregon se megjithëse në vlera numerike qendërsia e nyjeve shfaqet e njëjtë në strukturën e rrjeteve të Parisit, pozicionimi i nyjeve me qendërsi të lartë në kohë është i ndryshueshëm. Ai bën një krahasim në kohë të fazave të zhvillimit të qytetit, duke marrë në konsideratë periudhat e planifikuara dhe ato vetë organizuese. Në këtë studim, qendërsia e nyjeve do të trajtohet më tepër si vizualizim i hartave të qyteteve, por edhe si diskutim në nivel sasior.

3.4.4 Orientimi i rrugëve

Rrjetet në një qytet, organizojnë dinamikën njerëzore të sistemeve komplekse urbane. Ato orientojnë sjelljen e udhëtimeve, vendndodhjet dhe strukturën urbane (Jacobs A. B., 1995). Në veçanti, orientimi i rrjetit dhe gjeometria e tij, që në zanafillën e planifikimit urban, kanë luajtur një rol të madh (Smith, 2007). Matja e orientimit të rrjetit të rrugëve, mund të ndihmojë planifikuesit të kuptojnë zanafillën e planifikimit urban dhe morfologjinë e rrjetit, për të vlerësuar modelet ekzistuese dhe eksplorimin e metodave të reja për to. Orientimi më i zakonshëm në mbarë botën, madje edhe midis qyteteve që nuk kanë një rrjet të fortë, priret drejt veri-jug-lindje-perëndim.

Kohët e fundit, studiuesit kërkojnë mbi rregullin dhe kaosin nëpër qytete si atribut të qarkullueshmërisë dhe orientimit të rrjeteve. Prandaj, i kushtohet shumë vëmendje konfigurimit të rrjetit të rrugëve (Batty M. , 2005); (Boeing G. , 2017); (Marshall S. , Streets and Patterns, 2005); (Massuci, D, A, & M, 2009). Rrjetet rrugore mund të jenë të projektuara sipas principeve bazë të planifikimit urban, ose ato mund të jenë të zhvilluara organikisht si nevoja të adaptimit të mënyrës së jetesës. Konfigurimi i rrugëve dhe orientimi i tyre na ndihmon të përcaktojmë rregullat hapësinore dhe logjikën e zhvillimit të qyteteve. Në këto rrjete hapësinore, entropia ka lidhje të thella teorike me kompleksitetin (Batty M. , 2005). Një grup studiuesish kanë eksploruar natyrën e entropisë dhe rendin në rrjetet e rrugëve urbane, duke kërkuar të përcaktojnë modele të rendit dhe kaosit në sistemet urbane (Gudmundsson & Mohajeri, 2013); Kohët e fundit, studiuesit kanë studiuar rendin dhe çrregullimin e rrjetit të rrugëve përmes entropisë së orientimit. E para mat lakimin e rrugëve dhe si lidhet kjo me modelet dhe proceset e tjera urbane (Boeing G. , 2020); (Giacomin & Levinson, 2015); (Levinson & El-Geneidy, 2009).

Në këtë studim, në do të përcaktojmë dhe vizualizojmë entropinë e orientimeve të rrugëve për vlerësuar se sa të renditura janë ato (Gudmundsson & Mohajeri, 2013), pasi entropia përcakton konceptet e lidhura thelbësisht të çrregullimit, pasigurisë dhe shpërndarjes. Louf dhe Barthelemy (2014) eksplorojnë gjeometrinë e blloqeve të qytetit në mbarë botën si funksion i madhësisë së bllokut dhe faktorit të formës, grupimi i tyre për të identifikuar dallimet midis qyteteve të SHBA-së dhe Evropës. Megjithatë, dihet më pak për tendencat në orientimin dhe renditjen hapësinore të rrjeteve të rrugëve në mbarë botën. Në figurën 30, prezantohen vizualisht dy skema të rrjeteve rrugore dhe histogramat respektove, ku vërehen rrjeti ortogonal i orientuar veri-jug dhe lindje perëndim në të majtë dhe rrjeti organik, pa ndonjë orientim specifik në të djathtë.

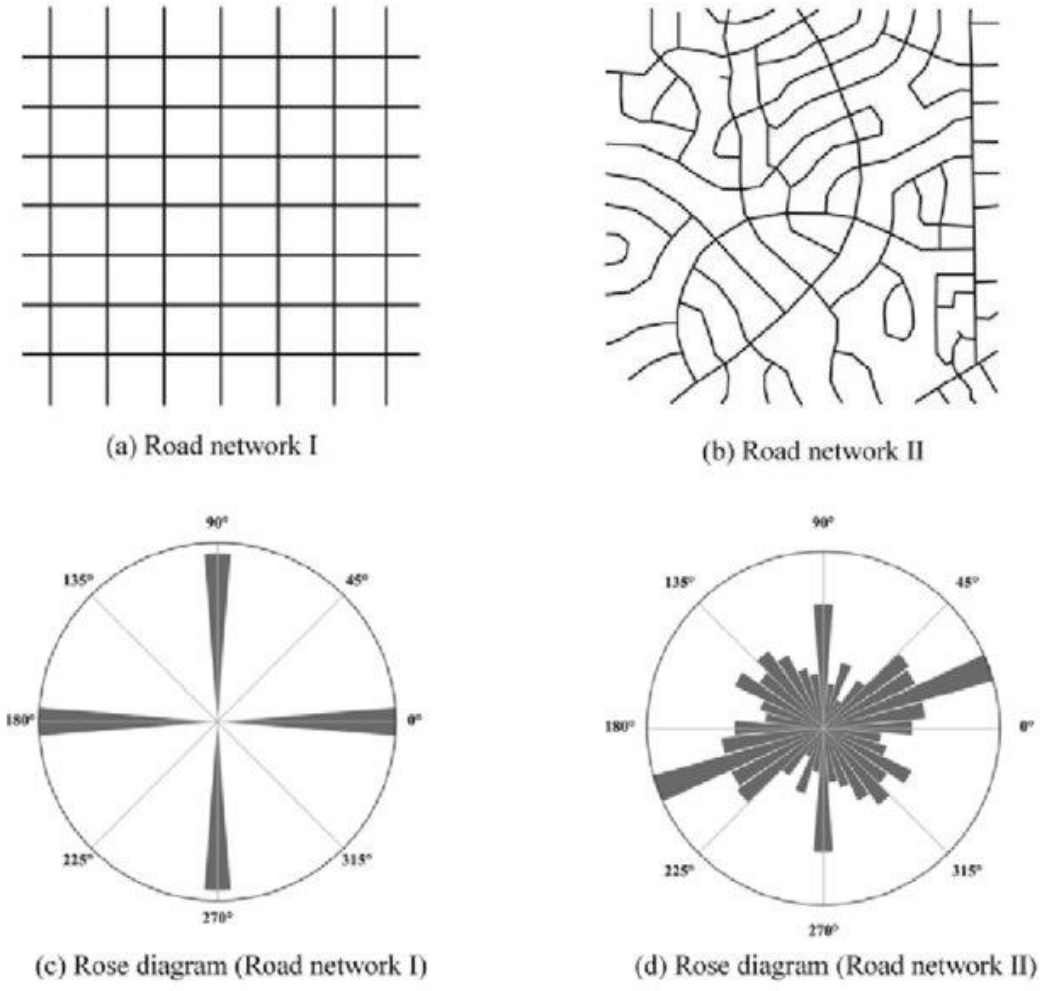


Figure 30 Dy skema të rrjetit rrugor dhe histogramat e tyre. Burimi: (Boeing G. , 2019)

3.5 Mbledhja e të dhënave

Krijimi i bazës së të dhënave për eksperimentin, do të realizohet në dy nivele.

Së pari, në nivel qytetesh, ku janë përzgjedhur tre qytete që kanë morfologji të ndryshme si të krijimit ashtu edhe të zhvillimit në kohë: Tirana, Fieri dhe Korça. Dhe së dyti në nivel kampionësh urbanë, siç u dakortësua mbi madhësinë e tyre në kapitullin e përzgjedhjes së kampionëve.

Inputi kryesor, që përbën bazën e të dhënave janë rrjetet, të cilat do të gjenerohen nga *OpenStreetMap*. Mjedisi urban i të tre këtyre qyteteve është i azhurnuar mjaftueshëm, për ti shërbyer qëllimit të këtij eksperimenti. Fillimisht, prezantohen këto qytete, nga një këndvështrim i shkurtër dhe perspektiva historike e zhvillimit urban të tyre. Më pas hulumton mbi vizualizimet dhe rezultatet krahasuese sasiore të strukturave të rrjetit. E njëjta analizë realizohet në nivel kampionësh dhe produktet grupohen në mënyrë krahasimore, për të arritur në konkluzionet e synuara.

3.5.1 Tirana, tiparet urbanistike dhe rrjeti rrugor

Sipas arkeologëve dhe një studimi të kryer nga (Mëhilli, 2014) Tirana është e banuar që në periudhën neolitike. Dhe kjo është vërtetuar nga studime të ndryshme, por dhe nga prania e shumë elementeve arkeologjike të trashëguara, si relike apo mbetje kishash, xhamish, fortesash e kështjellash. Që nga krijimi i qytetit të Tiranës vihet re karakteri tregtar i saj, duke qenë në një pozicion kyç, në kryqëzimin e rrugëve rajonale tregtare, konkretisht Shkodër - Elbasan; Kavajë – Dibër dhe rruga tregtare që lidh qytetet e Durrësit – Shëngjinit; i cili shërbente edhe si aks i rëndësishëm rreth të cilit përcaktohej qyteti. Procesi i zhvillimit në qytetet tregtare është më i lehtë dhe më i shpejt në krahasim me vendet e tjera, njëkohësisht edhe një nga arsyet kryesore pse Tirana u bë shumë shpejt një qytet i rëndësishëm.

Me ndërtimin e lagjeve të para, lagjja Bami, Mujo dhe Pazari i Vjetër (Mëhilli, 2014), u zhvillua infrastruktura si dhe sistemi i banimit ku secili prej tyre kishte një hapësirë private dhe rivitalizimi i lumit të Tiranës. Në vitin 1789 u ndërtua një nga elementët më të rëndësishëm të trashëgimisë kulturore të qytetit – xhamia e Haxhi Et'hem Beut. Gati pas gjysmë shekulli, në vitet 1821-1822 mbaroi ndërtimi i Kullës së Sahatit. Këto ndërtime janë monumentet më të rëndësishme të ditëve të sotme dhe disa nga atraksionet më të vizituara të qytetit.

Në shkurt të vitit 1920 Tirana u shpall kryeqyteti i Shqipërisë, jo për sfondin e tij historik, karakterin apo për monumentalitetin si qytet. Por kishte pozicionin gjeografik mjaft interesant, distancën e shkurtër të qytetit me detin dhe malin; praninë e shumë elementeve natyrore dhe burimeve ujore; tokën pjellore e cila mund të përdorej për zhvillimin dhe rritjen e popullsisë, si dhe pozita strategjike që mund të zgjeronte aktivitetet tregtare dhe zhvillimin ekonomik.

Me ndryshimet politike dhe me Ahmet Zogun si Mbret i Shqipërisë, qyteti filloi të përjetonte ndryshimet e para në strukturën dhe identitetin e tij urban. Në vitet 1917-1918 Tirana u projektua fillimisht në një hartë nga studiues austriakë, e paraqitur më sipër.

Pushteti totalitar është shprehur përgjithësisht përmes një planimetrie boshtore në strukturën urbane; ideja për të ndjekur një aks, apo një bulevard, të çon në një vend i cili është i rëndësishëm dhe tregon një fuqi të pamohueshme. Roma është shembulli i përsosur i një qyteti boshtor, nëse ndiqet rruga Via Della Conciliazione[33], mbërrihet në hyrjen e Katedrales së Shën Pjetrit, e cila të çon drejtpërdrejt në altarin e Katedrales (Elezaj & Fleury, 2015). Po kështu edhe Tirana është një tjetër qytet që mund të merret si shembull i një qyteti boshtor, pasi është projektuar me një ideologji italiane (figura 31).

[33] Rruga drejt Pajtimit

Figure 31 Harta e Tiranës 1917, nga arkitektë dhe inxhinierë Austriakë. Burimi: AQTN (TR030, 2016)

Menjëherë pas përfundimit të Luftës së Dytë Botërore, e gjithë vëmendja e qeverive u kthye në rindërtimin e vendeve; në Shqipëri, në Kongresin II të PPSH-së u vendos si objektiv kryesor që vendi të kthehet nga një vend bujqësor në një vend bujqësor-industrial. Transformimet e para në Tiranë nisën rreth vitit 1947, me ndërtimin e objekteve të ndryshme industriale si Kombinati i Mishit, Kombinati i Qumështit, Fabrika e Këpucëve dhe Kombinati më i famshëm i Tekstileve. Në vitet 1957-1958 u propozua një plan i ri rregullues për qytetin e Tiranës, bazuar në idetë e vitit 1953 të projektuar nga arkitekti Gani Strazimiri dhe në planin e hartuar gjatë periudhës fashiste, i cili pati një ndikim të rëndësishëm në strukturën urbane të qytet.

Kishte një diferencim të thellë midis qendrës së qytetit, e cila përdorej si element monumental dhe solemn i kryeqytetit dhe vëmendjes ndaj tij (Pojani & Boussauw, 2014) dhe periferisë, prandaj ndërtesat industriale projektoheshin dhe ndërtoheshin në periferi pranë tokave bujqësore; si funksione që e ndihmuan qytetin të lulëzonte.

Për më tepër, qeveria e asaj kohe kontrollonte procesin e urbanizimit dhe kapacitetin e popullsisë në qytet, për këtë arsye Plani i Ri Rregullues propozoi blloqe pallatesh jo më të larta se 5 kate. Disa ndërtesa banimi të këtij lloji janë pallatet në rrugën Bajram Curri, në rrugën e Konferencës së Pezës, rrugën e Durrësit, kompleksi i banimit në Kombinat, kompleksi i Shallvareve dhe 9-katëshet (pallatet më të larta të banimit të pranishme në qytet në atë periudhë, para ndërtimit të Hotel Tirana). Plani i Ri Rregullues propozoi gjithashtu standarde të reja për ndërtimin e rrugëve, unazave të Tiranës, përmirësimin përgjatë disa segmenteve të lumit Lana si dhe ndërtimin e një stacioni të ri treni.

Pas rënies së komunizmit në Shqipëri, dhe krijimit të një sistemi të ri me qasje pluraliste-demokratike, pasuan shumë ndryshime në sektorin ekonomik, në aspektin fizik të qyteteve, në aspektin social dhe në reformën territoriale. Vendi kaloi nga një sistem qendror dhe i kontrolluar në një gjendje liberale zhvillimi të pakontrolluar.

Procesi i urbanizimit u zhvillua në një mënyrë shumë të shpejtë dhe të paplanifikuar, në ndryshim nga periudha socialiste, duke shkaktuar shtrirjen urbane në periferi të qytetit, ku më parë kishte vetëm zonat industriale dhe bujqësore; ku funksioni u shndërrua kryesisht në funksion banimi. Nga ky fenomen u prekën edhe zonat urbane, duke prishur ekuilibrin e siluetës vertikale të ndërtesave të mëparshme të banimit të ulëta me ndërtesa të reja më të larta.

Fillimi i vitit 2000 shënoi fillimin e ndërgjegjësimit social dhe administrativ nëpërmjet reformave të ndryshme qeveritare; duke nënkuptuar ndërhyrjet në qendër të qytetit, rehabilitimin e lumit Lana dhe heqjen e vendbanimeve informale, zhvillimin e aktiviteteve të ndryshme tregtare në segmentin që lidh Tiranën me Durrësin etj. Megjithatë, ndjenja e kaosit nuk është zhdukur; ka ende një lëvizje ekstreme njerëzish drejt Tiranës në kërkim të mundësive të reja.

3.5.2 Fieri, tiparet urbanistike dhe rrjeti rrugor

Qyteti i Fierit shtrihet në pjesën perëndimore të Shqipërisë, në jug të fushës së Myzeqesë dhe ndodhet 18 km larg bregut të Detit Adriatik. Fieri ndodhet në koordinatat gjeografike midis 40°57'05" gjerësi gjeografike në veri, 40°33'06" gjatësi gjeografike në lidhje dhe 19°18'29" në vijën bregdetare në perëndim të Shqipërisë. Pozicioni gjeografik rajonal i Fierit është tepër i favorizuar për zhvillimet socio-ekonomike dhe strategji të zhvillimit pasi në të kalojnë dy nga korridoret më të rëndësishme të vendit të cilët janë Korridori VIII dhe Korridori i Kaltër (Autostrada Adriatiko-Joniane). “Qyteti është themeluar në 1694, nga Kahreman Pasha Vrioni dhe është rindërtuar sipas sistemit modern, në 1877 nga Omer Pasha Vrioni”. Në të njëjtën kohë, ky fshat i madh (Fier Qebir, sipas dokumenteve të Sharias së Berati) nuk ishte pronë e tij, por feudali i Sulltan Abdyl Azis. Ky fakt provohen nga gjurmët otomane në qendër të qytetit. Një nga qendrat historike më të rëndësishme në Ballkan, Apolonia e cila mendohet te jetë ndërtuar në vitet 600 para Krishtit, ka qenë një pikë e rëndësishme tregtare dhe ekonomike të kohës.

Ndërkohë, themelimi i qytetit të Fierit lidhet me rënien e Voskopojës si qendër urbane. Në themelimet e qytetit, popullsia ka qenë e vogël, por gradualisht është rritur në proporcion me rritjen dhe zhvillimin e Fierit, i cili fal pozicionit gjeografik si një kryqëzim i rrugëve lidhëse të qytetit me provincat e tjera, ishte në zhvillim të vazhdueshëm. Fillimi I shekullit XX e gjen Fierin me një fizionomi më të përmbushur. Udhëtuesja angleze Edith Durham, e cila e vizitoi qytetin në 1904, ishte e impresionuar nga bukuritë, gjallëria, rrugët dhe njerëzit e këtij qyteti.” … Ndërtesa prej guri solid janë ngritur,” shkruan ajo, “me një arkitekturë çuditërisht moderne.”

Në vitet 1960 qyteti u bë një qytet industrial dhe një nga bazat kryesore të prodhimit në vend të industrisë kimike dhe naftës me afërsisht 12% të prodhimit në të gjithë vendin. Agrikultura, edukimi, shëndetësia, kultura, sportet, etj. patën një rritje të konsiderueshme.

Rënia e sistemit komunist në vitet 1990, ashtu si në gjithë Shqipërinë edhe në Fier, solli ndryshime të mëdha në aspektin ekonomik, territorial dhe mjedisor. Pas viteve '90, pati një dobësim të industrive dhe zhvillimit të sektorit të tregtisë dhe atij të ndërtimit. Gjithashtu, në vitet e para të pas komunizmit u vu re një rritje e shpejt e popullsisë e si pasojë një zgjerim i shpejt i qytetit. Sot, Bashkia e Fierit gëzon një pozicion strategjik në territorin shqiptar i mundësuar nga disa avantazhe ndër sektoriale. Ajo përbën 5.7% të tokës bujqësore; kufizohet nga dy lumenj kryesor, Semani në veri dhe Vjosa në jug, të cilët kanë sipërfaqe të mbrojtura natyrore pranë grykëderdhjeve të tyre; zotëron Parkun Arkeologjik Kombëtar të Apolonisë; trashëgon zona industriale dhe naftëmbajtëse si dhe gëzon pozicion strategjik për sa i përket gazifikimit të mundshëm, me vënien në zbatim të Gazsjellësit TAP që kalon brenda territorit të saj (PPV Fier, 2016).

3.5.3 Korça: tiparet urbanistike dhe rrjeti rrugor

Qyteti mesjetar i Korçës thuhet se e ka zanafillën nga kalaja. Megjithatë, mungojnë gjurmët e nevojshme për të vlerësuar saktë këtë hipotezë. Nga ana tjetër, për strukturën urbanistike të Korçës, ka mjaft dëshmi historike që na ndihmojnë të kuptojmë më tepër tiparet e këtij qyteti.

Nga shkrimet e autorëve si Karmici (1888), mësojmë se krahas 7 lagjeve të qytetit që janë ende sot pjesë e identifikueshme e territorit, mbetet edhe një lagje, lagja Kala, e cila mendohet se shtrihej buzë lumit të Moravës. Ndërkohë, nga një dokument i vitit 1631, e cila është një letër ankese drejtuar Portës së Lartë (Perandorisë Osmane) mësojmë se në shekullin XV-të, struktura urbanistike e Korçës përbëhej nga kalaja, qyteti i hapur dhe pazari. Qyteti i hapur përbëhej nga lagjet jashtë kalasë, që shtrihen në anën lindore. Kjo pjesë e qytetit deri vonë njihej me emrin Varosh. Pjesa e tretë e cila përben edhe bërthamën formues të qytetit ishte pazari. Pazari i vjetër mendohet se shtrihej pranë lagjes Penço (pjesë e Varoshit).

Struktura e qytetit të Korçës nisi të ndryshojë në shek XV-të me ardhjen në pushtet të Iljaz Bej Mirahorit. Ai zhvendosi dhe ndërtoi mjaft objekte me rëndësi të lartë shoqërore, fetare dhe ekonomike në fshatin e tij, Peshkëpi. Dhe jeta urbane e Korçës u lidh mjaft me këtë zonë, e cila më vonë u bë pjesë e qytetit e njohur si lagjja Kasaba. Deri në mesin e shekullit të XIX-të, Korça vijonte të ishte e ndarë në tre njësi të mëdha: lagjja Varosh (ose lagjja ortodokse), Kasabaja (lagjja myslimane) dhe pazari. Ndërtimet e objekteve të kultit nuk e kanë ndryshuar apo ndikuar fizionominë urbane të qytetit të vjetër deri në mesin e shekullit të XIX-të, kur nisi edhe zgjerimi i mëtejshëm i qytetit. Rreth kishës u formua një shesh, i cili luan edhe rolin e një qendre shoqërore dhe pikëtakimi për qytetin. Ky shesh ndikoi edhe në strukturën urbanistike të qytetit, pasi u rrethua me parcela të dendura të banuara dhe shumica e arterieve rrugore lidheshin në këtë shesh.

Rrjeti rrugor i qytetit të vjetër, i cili paraqitet i parregullt dhe mjaft gjarpërues, ruhet ende sot në një pjesë të mirë të lagjes së vjetër. Sipas Pirro Thomos (2022), kjo gjë dëshmon qartë për karakterin e lirë dhe mjaft spontan të zhvillimit urbanistik të qytetit. Banesat qytetare kryesisht 2 katëshe, me shumë dritare dhe kryesisht të pajisure me qoshk ose erker të dalë pezull mbi rrugë, janë një pjesë e rëndësishme e formulimit urbanistik dhe arkitekturor të qytetit, pasi na mundësojnë pamje mjaft dinamike. Gjerësitë e rrugëve janë zakonisht 2-4 m, sipas një dokumenti të mbledhur nga P. Pepo (1972), pasi gjerësia minimale duhej të ishte aq sa të

kalonte një *araba*[34]. Rrugët e rrugicat ishin kryesisht të shtruar me kalldrëm dhe me kanalet që ndihmojnë ne largimin e ujërave në të dyja anët e rrugëve.

Zhvillimi i qytetit të Korçës kishte arritur një shkallë të lartë në pjesën e parë të shekullit të XIX-të. Ndryshimet dhe dinamikat urbanistike ne lagjet e vjetra të qytetit sollën edhe ndryshime të karakteristikave që përbënin bërthamën e këtij qyteti. Shtimi i numrit të popullsisë dhe zhvillimi i madh ekonomik për kohën ndodhi në vitet 1867-1873, dhe kjo dokumentohet nga akte të shumta shitjeje të tokave e trojeve, si edhe nga datat e banesave të ndërtuara gjatë kësaj periudhe. Edhe pse zgjerimi i Korçës erdhi me faza, veç krijimit të lagjeve të reja ndodhi edhe dendësimi i ansambleve të vjetra. Qyteti fitoi një strukturë më të rregullt urbanistike gjatë kësaj periudhe dhe kjo dëshmohet nga sistemi pothuajse ortogonal, me disa përjashtime nga rregullat strikte gjeometrike, pasi sigurisht u desh që zhvillimi dhe zgjerimi t'i përshtatej kushteve të terrenit dhe klimës së qytetit (figura 32).

Nga mesi i shek. XIX-të e deri në fillimin e shek. XX-të, zgjerimi i qytetit të Korçës u bë në faza. Gjatë fazës së parë erdhi zgjerimi në verilindje i bërthamës së Varoshit deri pranë kodrave të Shën Thanasit. Ndërsa në fazën e dytë, zgjerimi u shtri në trojet veriore. Gjatë kësaj faze kemi edhe shtimin e një aksi, rrugën e Follorisë, që sot njihet si bulevardi Republika, i cili përshkon qytetin mes për mes. Degëzimi i kësaj rruge e ndan qytetin në anën veriperëndimore, duke krijuar një formë unazore që përfundon me tregun e qytetit. Lagjet e reja të qytetit kanë një rrjet më të rregullt e të dendur rrugor, pasi rrugët ndërpriten ne kënde të drejta. Parë nga lart, kjo ka sjellë krijimin e parcelave të vogla drejtkëndore me sipërfaqe rreth 200-500 m^2. Në pjesën e parë të shekullit të XX-të, krahas krizës ekonomike botërore dhe pushtimit të Shqipërisë nga Italia fashiste, kemi edhe industrializimin e ngadaltë të vendit. Në Korçë, kjo fazë u shfaq në qytet me ndërtimin e disa fabrikave të vogla (të leshit, tullave, miellit, alkoolit, duhanit, birrës (1928) etj.).

Ndryshimi i qytetit në vitet pasardhëse u ndikua nga disa vendimmarrje politike dhe ekonomike, si tharja e kënetës së Maliqit, devijimi i përroit të Korçës, plani rregullues i qytetit, pyllëzimi i kodrave e maleve, etj. Ndërtimet e reja në qytet ishin me ritme të kënaqshme gjatë kësaj periudhe, por mungonin fondet për investime të mira komunale e infrastrukturore. Kërkesa për hapësira banimi në qytet u reflektua me ndërtimin e mjaft objekteve që kryesisht zinin vend në lagjet ekzistuese duke zëvendësuar objektet ekzistuese, ose duke dendësuar më tej lagjet e vjetra. Ndërkohë, pjesa e re e qytetit u ndërtua sipas skemës urbanistike që ishte përdorur edhe më herët: zgjerimi në pjesën veriore të qytetit dhe u shtua epërsia e sistemit ortogonal.

Plani rregullues i qytetit të Korçës mbetet ende një pjesë e pasigurt për shkak të mungesës së dokumentacionit. Megjithatë, zgjerimi i qytetit sipas rrjetit ortogonal për gati gjysmë shekulli, na dëshmon se supozimi për një akt apo plan të qytetit qëndron. Megjithatë, në vitin 1920, kemi njoftimin e parë për hartimin e një plani urbanistik, ku sipas artikullit "Karta e qytetit Korçë", botuar në Gazetën Korça në 20 maj 1910, mësohet se parashikohej zgjerimi i rrugëve të vjetra me gjerësi 8-12 m, sistemimi i lagjeve të vjetra, etj. Një dekadë më vonë, ngrihet kërkesa për një plan rregullues, për shkak se qyteti po zgjerohej me shpejtësi. Në vitin 1928, nis hartimi i

[34] qerre

planit nga Kohler dhe Bertoldi, i cili u finalizua në vitin 1930. Plani rregullues e propozon Korçën mjaft të formuar si territor, si strukturë urbane dhe në rrugën e zhvillimit të mëtejshëm të koncepteve urbane ekzistuese në territor. Nis me planin edhe rregullimi i mënyrës së ndërtimit, duke disiplinuar zhvillimin me anë të lejeve të ndërtimit. Gjatë një krahasimi të planit me gjendjen ekzistuese të qytetit sot, vërehet se vetëm një pjesë e vogël e propozimeve u zbatua në vijimësi. Kryesisht u vunë në zbatim disa zgjerime rrugës, shtimi i disa parcelave të reja për ndërtim, etj.

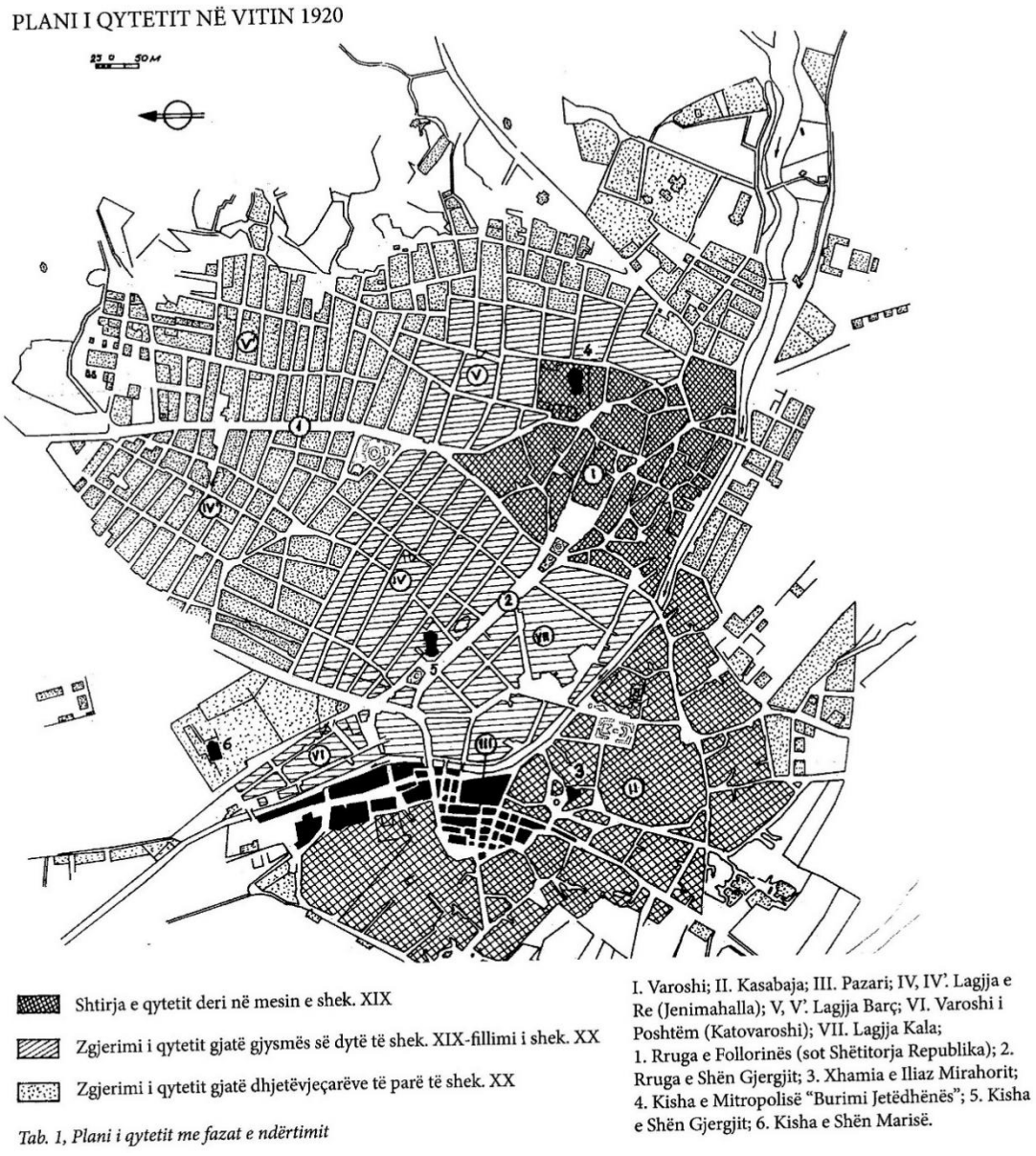


Figure 32 Plani i qytetit të Korçës me fazat e ndërtimit. Burimi: (Thomo, 2022)

REZULTATET E KËRKIMIT 4

4. REZULTATET E KËRKIMIT

4.1 Analizë krahasimore në nivel qytetesh

Në këtë kapitull, ne synojmë të shqyrtojmë dhe analizojmë rezultatet e arritura nëpërmjet softit në nivel qytetesh fillimisht dhe më pas në nivel kampionësh urbanë. Tabela e mëposhtme, në mënyrë të përmbledhur, paraqet të dhënat metrike dhe strukturore për tre qytetet e përzgjedhur. Në anekset 1 deri në 3, janë gjenerimet e softit respektivisht për qytetin e Tiranës, Fierit dhe Korçës.

Table 4 Tabela e të dhënave metrike dhe strukturore gjeneruar nga OSMnx, për tre qytetet, Tirana, Fier, Korça. Burimi: autori

Llojet e matjeve	Të dhënat	Qyteti_Tiranë	Qyteti_Fier	Qyteti_Korçë
Të dhënat të gjenerura nga OSMnx	Sipërfaqja (km^2)	68.35	9.72	12.09
	n (numri i nyjeve)	14,911.00	1,425.00	1,658.00
	m (numri i skajeve)	33,779.00	3,536.00	4,352.00
	Gjatësitë totale të rrugëve (m)	988,113.30	145,281.59	139,246.28
	Gjatësitë totale të skajeve (m)	1,774,651.61	280,675.72	261,743.15
	Numri i kryqëzimeve	10,727.00	1,101.00	1,431.00
Densiteti (Të dhëna metrike)	Mesatarja e gjatësive së rrugëve (m)	52.65	78.19	58.50
	Mesatarja e gjatësive të skajeve (m)	52.53	79.38	60.14
	Densiteti i nyjeve për km^2	218.15	146.62	137.10
	Densiteti i kryqëzimeve për km^2	156.93	113.28	118.33
	Densiteti i skajeve për km^2 (m)	25,962.85	28,879.23	21,643.74
	Densiteti i rrugëve për km^2 (m)	14,455.92	14,948.29	15,617.54
	Kryqëzime me 2 rrugë	46.00	-	3.00
	Kryqëzime me 3 rrugë	9,205.00	951.00	1,143.00
	Kryqëzime me 4 rrugë	1,450.00	147.00	281.00
	Kryqëzime me 5 rrugë	25.00	1.00	3.00
	Kryqëzime me 6 rrugë	1.00	2.00	1.00
Lidhshmëria (Matje strukturore)	Qarkullueshmëria mesatare	1.07	1.05	1.04
	Shkalla mesatare e nyjes	4.53	4.97	5.24
	Mesatarja e rrugëve për nyje	2.53	2.65	2.90
	Densiteti i nyjeve për km^2	218.15	146.62	137.10
	Densiteti i kryqëzimeve për km^2	156.93	113.28	118.33
	Raporti i laqeve të mbyllura	0.0027	0.0016	0.004
Qendërsia (Matje strukturore)	Qendërisa e nyjeve	0.15	0.19	0.233

Siç shihet në rezultatet e matjeve, fillimisht jepen të dhëna të përgjithshme të gjeneruara nga softi pas thjeshtimit strukturor që u ndodh rrugëve të qytetit, në një graf joplanar, të drejtuar. Të dhënat e gjeneruara nga *OpenStreetMap*, për të tre qytetet lidhen me hapësirën urbane të tyre, para reformës territoriale të vitit 2015. Për qëllimin e studimit tonë, kjo vlen pasi është hapësirë më e azhurnuar në pikëpamjen e rrjeteve në *OpenStreetMap*. Siç, vërehet, të tre qytetet kanë sipërfaqe konsiderueshëm të ndryshëm nga njëri tjetri, ku Tirana është rreth 6-7 herë më e madhe si sipërfaqe sesa qyteti i Fierit, apo i Korçës. Fieri dhe Korça kanë sipërfaqe të krahasueshme me njëri tjetrin.

Për nga origjina dhe natyra e zhvillimit të tyre, të tre qytetet shfaqin karakteristika të ndryshme të zhvillimit hapësinor të tyre, dhe sidomos të strukturës së rrjeteve, të lidhura këto me gjenezën e formimit të tyre dhe presioni për zhvillim në kohë dhe në hapësirë. Të dhënat që gjenerohen,

ndahen në tre grupime, për të thjeshtuar analizën e rrjetit në tregues që lidhen me dendësinë e rrjetit, lidhshmërinë dhe qendërsinë e tij. Megjithatë këta tregues ndërlidhen me njëri tjetrin për të zhvilluar një këndvështrim më të plotë. Orientimi i rrjetit rrugor është tregues i qëndrueshmërisë dhe gjenezës së krijimit të qyteteve. Gjithashtu, janë realizuar gjenerime vizuale të rrjetit (figura 33) për të kuptuar grafikisht strukturën e rrjetit dhe hapësirën urbane të marrë në studim.

4.1.1 Densiteti

Në terma të densitetit analiza do të zhvillohet duke krahasuar:

(i) Mesataren e gjatësive të rrugëve
(ii) Densitetin e kryqëzimeve për km^2
(iii) Densitetin e rrugëve për km^2
(iv) Shpërndarjen e rrugëve sipas numrit të kryqëzimeve

Figure 33 Vizualizimi i rrjeteve pas thjeshtimit përmes OSMnx, përkatësisht Tiranë, Fier, Korçë. Burimi:autori

Dendësia e kryqëzimeve është e lidhur ngushtë me madhësinë e bllokut apo me mesataren e gjatësive të rrugëve dhe rrjedhimisht me shkallën e qytetit. Në veprën Jane Jacobs (1961), ajo identifikon blloqet e shkurtra si një nga gjeneruesit e larmishmërisë. Blloqet më të shkurtra rezultojnë në më shumë kryqëzime dhe në fund në më shumë mundësi për njerëzit që të takohen. Rezultatet nuk janë surprizuese për të tre qytetet. Nëse shohim raportin e këtyre tre qyteteve, rezulton se Tirana ka mesataren e gjatësive të rrugëve më të vogël, më pas Korça me shifra të përafërta, dhe në fund Fieri. Kjo tregon që madhësia e bllokut në Fier është më e madhe se e dy qyteteve të tjera (figura 34).

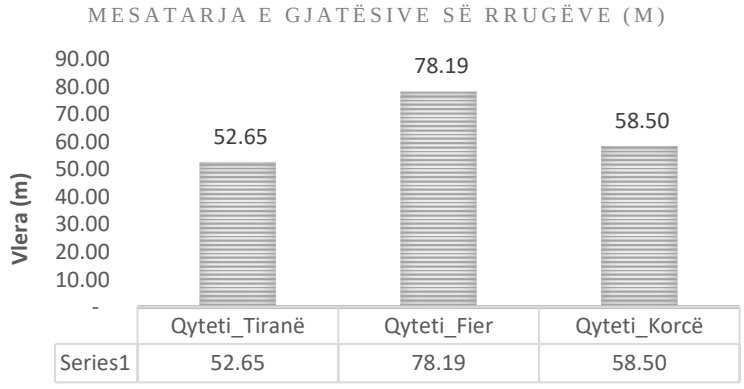


Figure 34 Diagrame krahasuese - mesatarja e gjatësive të rrugëve. Burimi:autori

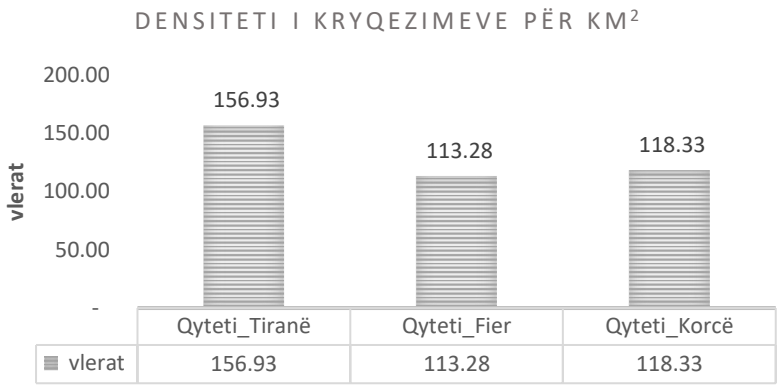


Figure 35 Diagrame krahasuese - densiteti i kryqëzimeve për km². Burimi:autori

I njëjti raport ndiqet edhe për sa i përket densitetit të kryqëzimeve për km², ku rezulton se Tirana ka numër më lartë kryqëzimesh për km² (figura 35). Nëse shohim shpërndarjen e numrit të rrugëve sipas numrit të kryqëzimeve, vërehet në të tre qytetet që mbizotërojnë qytetet me rrugë me tre kryqëzime, e ndjekur me ato me 4 kryqëzime. Për shkak të shtrirjes më të madhe rezulton që Tirana e ka disa herë më të madh numrin e kryqëzimeve me tre rrugë (figura 37).

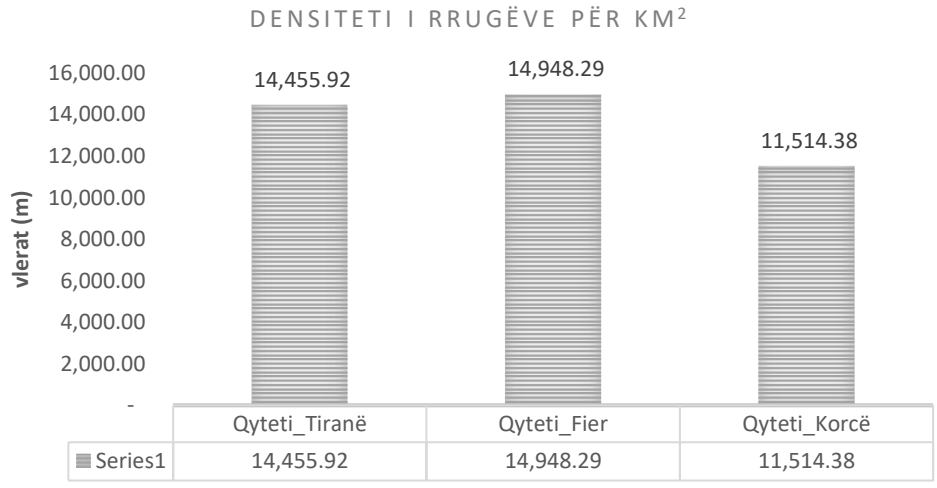


Figure 36 Diagrame krahasuese- densiteti i rrugëve për km². Burimi: autori

Densiteti i rrugëve për km², është një tregues që lidhet si me densitetin e rrjetit ashtu edhe me lidhshmërinë e tij dhe qartësisht flet për një fragmentizim të rrjetit në Tiranë dhe në Fier (figura 36). Ky tregues nuk mund të shihet i pavarur nga densiteti i kryqëzimeve, apo mesatarja e gjatësive të rrugëve.

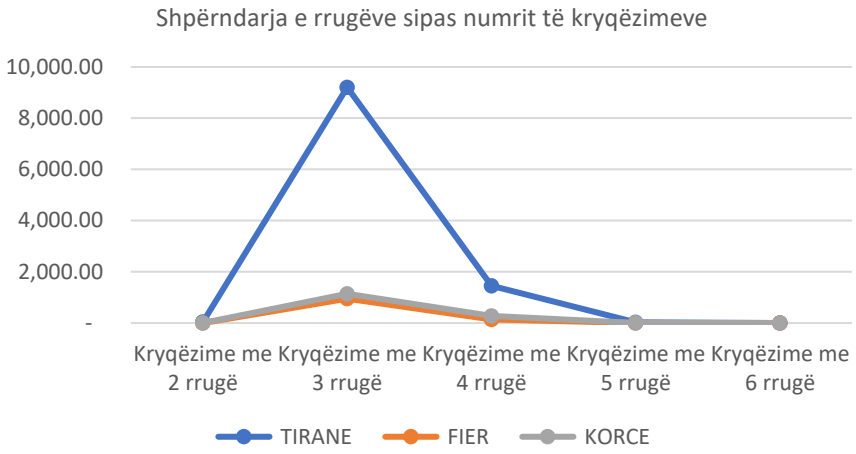


Figure 37 Diagrame krahasuese - Shpërndarja e rrugëve sipas numrit të kryqëzimeve. Burimi: autori

Në përfundim mund të themi që rrjeti i rrugëve paraqitet më i dendur në Tiranë, ku mesatarja e gjatësive të rrugëve, rrjedhimisht madhësia e bllokut paraqitet më e vogël në shkallë qyteti. Ky është tregues i një fragmentizimi të lartë të strukturës urbane. Vlen të theksohet fakti, që harta e Tiranës është me e azhornuar në *OpenStreetMap*. Megjithatë, pas Tiranës, vjen Korça e cila ka një bërthamë vernakulare të qytetit të krijuar shumë herët.

Përfundime:

Tirana paraqitet me rrjet të dendur dhe të imët, me madhësi të vogël të bllokut urban (rruga mesatare~53m). Kjo është tregues i presionit të madh të zhvillimit të viteve të fundit. Gjithashtu, edhe Korça paraqitet me rrjet të dendur dhe të imët, me madhësi blloku urban të vogël (rruga mesatare ~58m). Ndërsa Fieri paraqitet me rrjet më pak të dendur, dhe jo të imët, me madhësi blloku urban më të madhe (rruga mesatare ~78m).

4.1.2 Analizë e lidhshmërisë së rrjeteve

Është e qartë se matjet e lidhshmërisë janë shumë të larmishme, por ne do të përqendrohemi ne disa prej treguesve që lidhen me të. Në këtë kuptim, ne fokusohemi në parametrat si më poshtë:

(i) Shkalla mesatare e nyjes
(ii) Mesatarja e rrugëve për nyje
(iii) Qarkullueshmëria mesatare
(iv) Raporti i laqeve të mbyllura

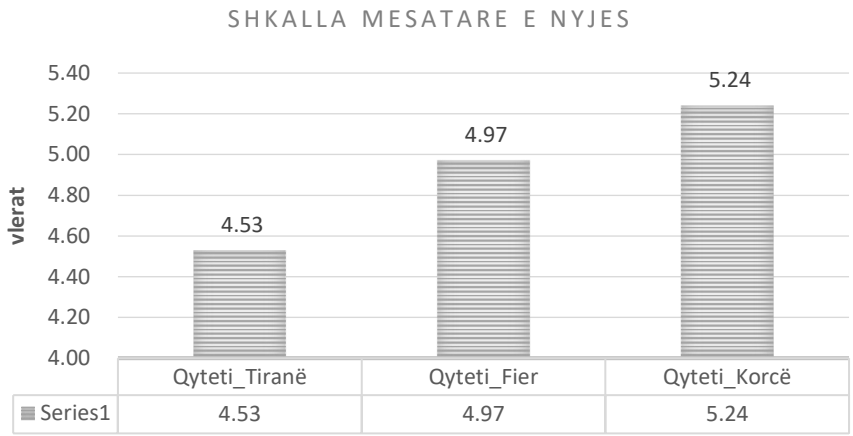


Figure 38 Diagrame krahasuese - shkalla mesatare e nyjes. Burimi: autori

Shkalla mesatare e nyjes është numri mesatar i skajeve që ndodhin në secilën nyje, përcakton se sa mirë janë të lidhura nyjet. Sa më e lartë të jetë shkalla mesatare e nyjes aq më i lidhur është rrjeti. Nëse krahasojmë tre qytete, rezulton se rrjeti në Korçë e ka më të lartë këtë indeks, çka tregon se ka lidhshmëri më të lartë (figura 38). Sipas këtij vlerësimi, Tirana ka lidhshmëri më të vogël, dhe Fieri rezulton në mes.

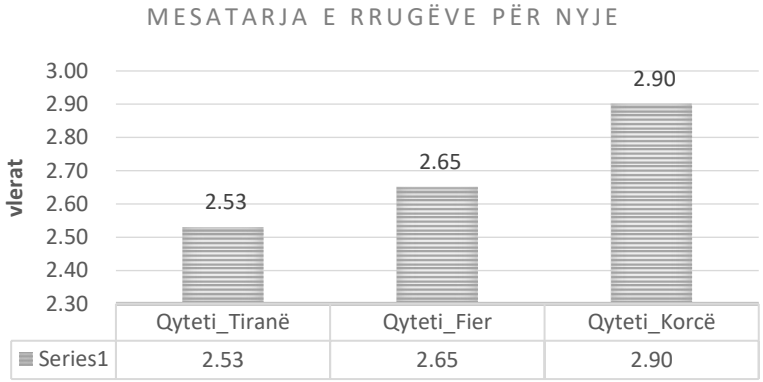


Figure 39 Diagrame krahasuese - mesatarja e rrugëve për nyje. Burimi: autori

Siç u cek edhe më sipër, tregues i llojit të rrjetit është edhe numri i rrugëve për nyje. Në qytet e planifikuara, si në Shtetet e Bashkuara të Amerikës, kjo shifër është rreth 4 (d.m.th. në një kryqëzim përfundojnë rreth 4 rrugë), çka do ishte idealja. Në diagramin e mësipërm, shohim që në Tiranë është 2.53, në Fier 2.65 dhe në Korçë 2.9, çka tregon se rrjeti në Korçë është më shumë i degëzuar dhe për rrjedhojë ka lidhshmëri më të lartë (figura 39).

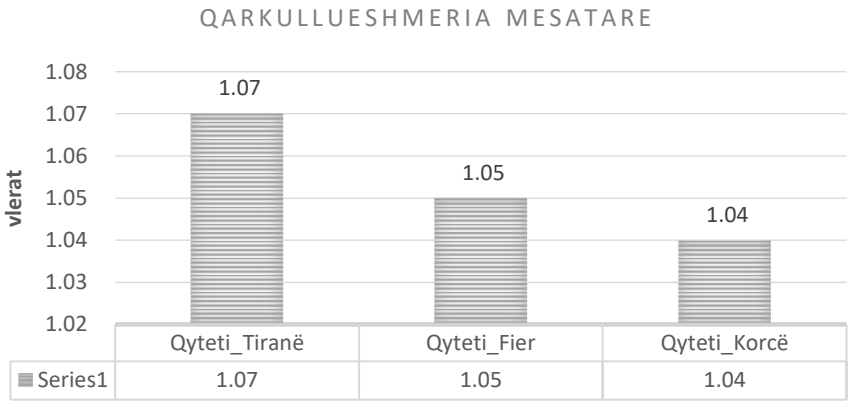


Figure 40 Diagramë krahasuese - Qarkullueshmëria mesatare. Burimi: autori

Edhe qarkullueshmëria mesatare, pavarësisht se ndryshimi është i vogël paraqitet në vlerë më afër vlerës 1, që është idealja në qytetin e Korçës (figura 40). Ky është raporti mesatar midis gjatësisë së një skaji dhe distancës së vijës së drejtë midis dy nyjeve që ajo lidh.

Raporti i laqeve të mbyllura është raporti i një skaji që lidhet vetëm në një nyje, qark i mbyllur që ka vetëm një skaj i cili fillon dhe mbaron tek vetja. Kuptohet që në nivel qyteti ky raport do të jetë shumë i vogël, dhe merr vlerë në nivel kampionësh. Ky është tipar i zhvillimeve të arkitekturës vernakulare. Ka vlerë më të lartë në qytetin e Korçës, pasi, është më i ndjeshëm në raport me qytetin në tërësi (figura 41).

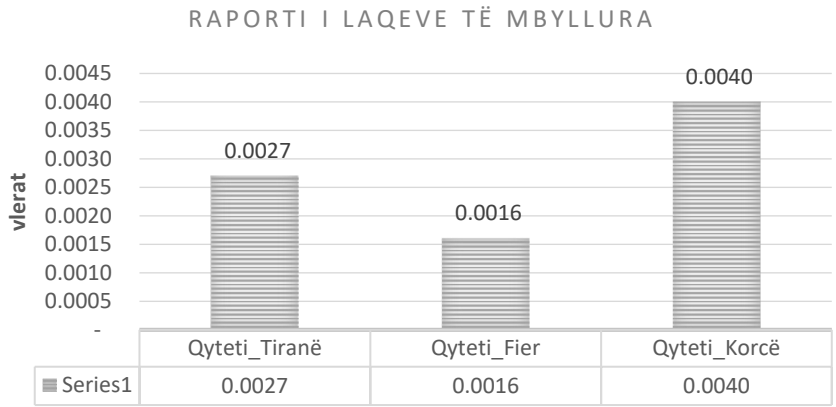


Figure 41 Diagrame krahasuese – Raporti i laqeve të mbyllura. Burimi: autori

Përfundime:

Në aspektin e lidhshmërisë, qyteti i Korçës paraqet rrjet urban me lidhshmëri më të lartë, më pas vjen qyteti Fieri dhe në fund kryeqyteti Tirana. Si përfundim arrijmë në konkluzionin se sa më e lartë është lidhshmëria, aq më e qëndrueshme paraqitet struktura urbane e qytetit.

4.1.3 Qendërsia

Në hartat e mëposhtme janë vizualizuar nyjet sipas qendërsisë nga më e errëta (vlera më e ulët) deri tek më e çelura (vlera më e lartë). Nëpërmjet softit të përdorur, vizualizohet se sa "e rëndësishme" është një nyje ose një skaj në një rrjet duke llogaritur qendërsinë e saj. Në shkencën e rrjeteve ekzistojnë shumë qasje për matjet e qendërsisë, duke përfshirë afërsinë, ndërmjetësinë, shkallën, vektorin e vet dhe PageRank.

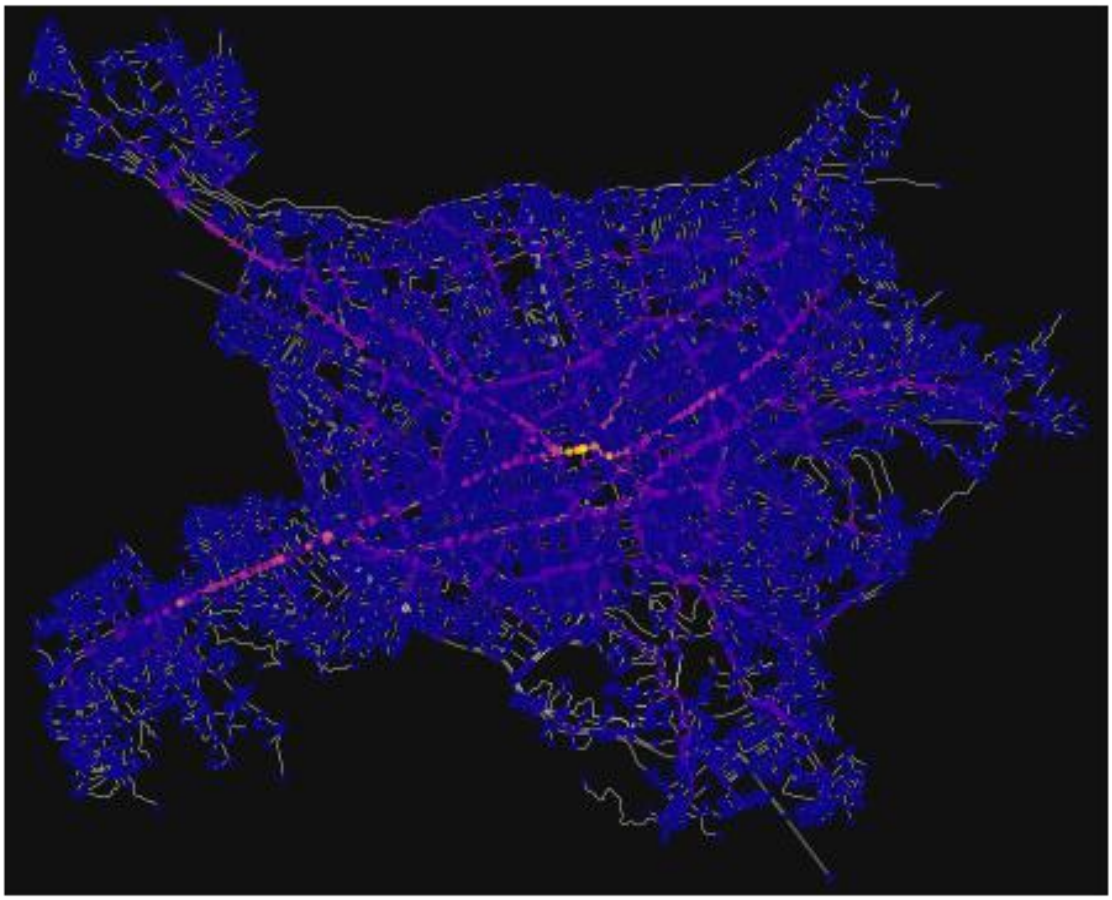


Figure 42 Vizualizimi i centralitetit të nyjeve, Tiranë. Burimi: autori

Nëse i vërejmë me kujdes për qytetin e Tiranës, nyjet me qendërsi më të madhe, janë pikërisht në akset kryesore të rrugëve. Nëse i referohemi të dhënave empirike të tabelës së mësipërme rezulton se 14,8% e rrugëve më të shkurtra kalojnë mesatarisht në një nyje në Tiranë (figura 42), 19% në Fier dhe 23.3% në Korçë. Kjo do të thotë që në Korçë ka më shumë nyje që përshkruhen nga rrugët më të shkurtra, çka i jep rendësi të madhe atyre dhe prishja e tyre, mund të shkatërrojë strukturën e rrjetit të qytetit. Kjo është karakteristikë e qyteteve me gjenezë vernakulare, për shkak të cikleve të adaptimit në kohë që kanë pësuar. Megjithatë vlerat nuk janë shumë të ndryshme nga njëra tjetra, për të krijuar një tablo të diferencueshme. Nëse analizojmë vizualisht hartën e Tiranës, nyjet kanë qendërsi më të lartë në unazat dhe akset kryesore, rrjedhimisht prishja/ humbja e kontrollit mbi to, do të thotë ngërce dhe joefikasitet.

Edhe në Fier, situata paraqitet e ngjashme, pasi qyteti ka karakter radial. Nyjet me qendërsi më të lartë janë pikërisht ato ku ka përqendrim më të madh të lëvizjes dhe fluks të lartë. Nëse

shohim vizualizimin për Korçën, rezulton se nyjet me qëndërsi më të lartë, janë përqendruar në një aks, dhe në ato nyje përfundojnë rreth 23.3% e rrugëve më të shkurtra (figura 43).

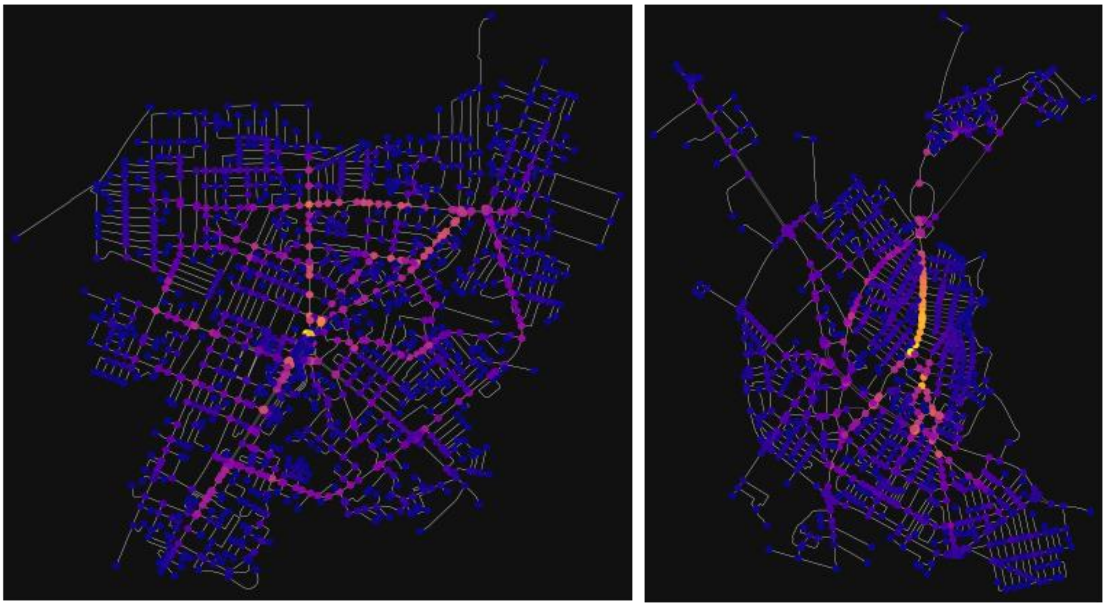


Figure 43 Vizualizimi i centralitetit të nyjeve, Fier dhe Korçë. Burimi: autori

Sipas Barthélemy et al. (2013), ai përdor analizat e qendërsisë për të identifikuar ndërhyrjet nga lart-poshtë kundrejt ndërhyrjeve poshtë-lart të vetë organizimit dhe evoluimit. Vlera më të ulta të qendërsisë, tregojnë për struktura të planifikuara, ndërsa vlera më të larta të qendërsisë tregojnë për ndërhyrje nga poshtë-lart dhe ciklin e ripërtëritjes që shfaqin qytetet. Korça është një shembull i tillë.

Qendërsia e skajeve Nëse i referohemi hartave të qendërisë së skajeve, për të tre qytetet, vizualisht prezantohen me ngjyrën e skajeve nga më e çelëta (qendërsi më e lartë) e deri tek më e errëta (qendërsi më e ulët). Rezulton se Tirana ka qendërsinë e skajeve më të lartë (ka arritshmëri më të lartë) sesa dy qytetet e tjera. Fieri e ka të përqëndruar në hapësirën e qendrës, ndërsa Korça ka të konsoliduar në një zonë të vetme (figura 44).

Përfundime: Në aspektin e qendërsisë, mund të gjykojmë që Korça ka vlerë të qendërsisë së nyjeve më të lartë, çka rezulton nga ndërhyrjet nga poshtë-lart dhe shfaqjen e ripërtëritjes adaptive si cikël të qytetit.

Figure 44 Vizualizimi i centralitetit të skajevE. Burimi: autori

4.1.4 Analizë krahasimore, orientimi i rrjeteve

Nuk është qëllimi i këtij studimi, zbulimi i atributeve kryesore që formojnë entropinë e orientimit. Në qëndrimet e mbajtura nga Joeff Boeing (2019), trajtohet në nivel treguesish dhe parametrash të tjerë zhvillimi i entropisë së orientimit të rrugëve. Teorikisht është trajtuar, në kapitullin e dytë dhe të tretë, të këtij disertacioni. Siç shihet, entropia e orientimit të rrugëve zhvillohet sipas një histogrami rrezor, i cili paraqet në formë vizuale orientimin e skajeve/rrugëve, sipas anëve të horizontit, duke vizualizuar gjatësinë si funksion i drejtimit (figura 45). Në aneksin 4, janë gjenerimet e softit për orientimin e rrjeteve për tre qytetet.

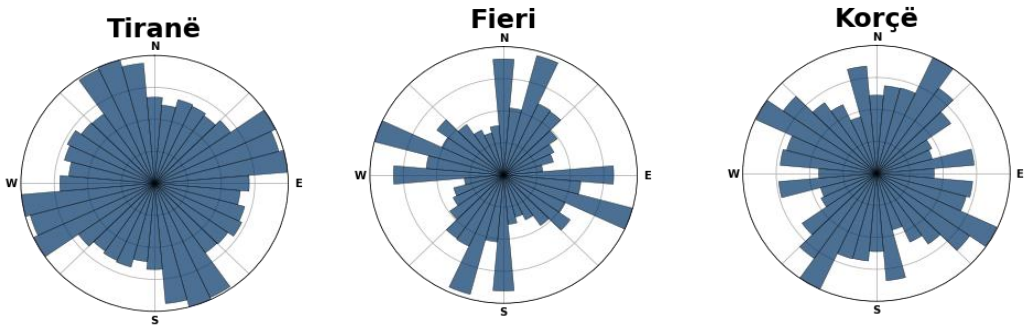


Figure 45 Histograma e shpërndarjes së orientimit të rrjeteve rrugore në qytete. Burimi: autori

Sipas histogramëve ne vërejmë që rrjeti rrugor i qytetit të Tiranës, paraqitet i ekuilibruar, ku vihen re dy drejtime të konsoliduara të zhvillimit të të rrjetit, pothuajse veri jug dhe lindje perëndim. Ky orientim e ka zanafillën që në bulevardin kryesor të Tiranës, i cili në mënyrë të vazhduar replikohet në të gjithë sipërfaqen urbane.

Zhvillimi i rrugëve edhe në periferi të saj ka një bindje gjeometrike që i ruan akset kryesore dhe në mënyrë komplementare përmbushet me rrugët e tjera pothuajse në të gjitha drejtimet. Një rrjet i tillë, flet për një artikulim të njëtrajtshëm në shpërndarje dhe përveç dy akseve që zënë peshën kryesore në qarkullimin e qytetit, rrugët e tyre drejtohen në mënyrë të barabartë në çdo drejtim. Vërehet një rend hierarkik në zhvillimin e rrjeteve të qytetit.

Qyteti i Fierit, siç shihet edhe në vizualizimin e rrjetit rrugor të gjithë qytetit, ka zhvillim radial brenda unazës së qytetit, bërthamës së vjetër urbane. Por duke qenë se ka pasur zhvillime informale pas viteve ’90, ato reflektohen në rrjetet e zhvilluara sidomos në pjesën veriore dhe perëndimore, ku zhvillohet një rrjet ortogonal, i ndryshëm nga pjesa tjetër. Kryesisht, rrjetet zhvillohen me orientimin veri jug dhe lindje perëndim, me një inklinim të vogël në verilindje jugperëndim. Histograma rrezore nuk paraqitet e ekuilibruar, ka drejtime të cilat janë të papërfillshme në skemën e përgjithshme të zhvillimit.

Nëse vlerësojmë histogramën e qytetit të Korçës, vërehet se janë dy akse shumë të rëndësishme verilindje-jugperëndim e veriperëndim-juglindje. Bërthama e vjetër e qytetit, e cila ka formën e gjysmë hënës, reflekton pikërisht këto dy akse dhe zhvillimi në kohë i qytetit, derivon prej tyre. Histograma paraqitet e ekuilibruar deri në një farë mase dhe ka hierarki në zhvillimin strukturor të rrjetit të qytetit. Përfundime: Tirana dhe Korça paraqiten me rrjet të ekuilibruar në aspektin e orientimit të rij, ndërsa Fieri paraqitet më kaotik.

4.2 Përzgjedhja e kampionëve urbanë, qyteti i Tiranës

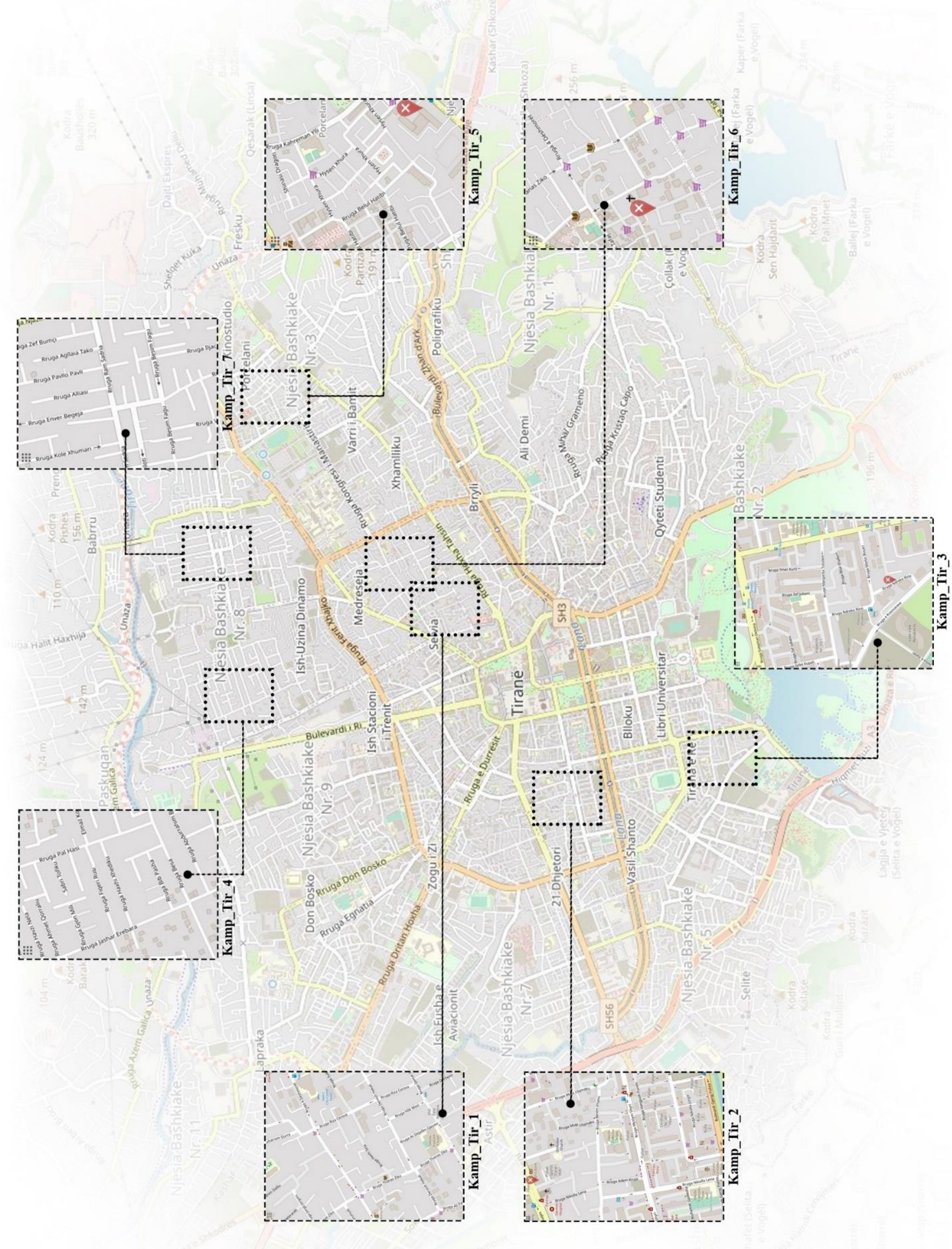


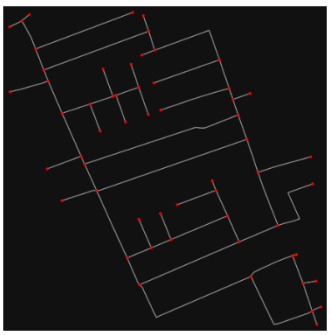
Figure 46 Përzgjedhja e kampionëve në qytetin e Tiranës. Burimi: autori

Tabela 5 Vizualizimi i kampionëve të rrjeteve përmes OSMnx, për Tiranën. Burimi: autori

Kamp_Tir_1

Kamp_Tir_2

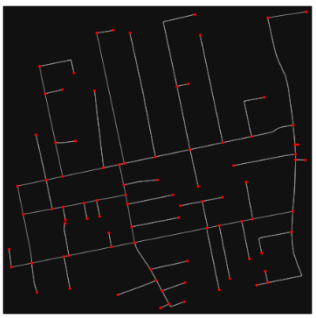
Kamp_Tir_3



Kamp_Tir_4

Kamp_Tir_5

Kamp_Tir_6



Kamp_Tir_7

Në tabelën e mësipërme paraqiten, sipas renditjes vizualizimet e rrjeteve, për kampionët e marrë në studim, pas thjeshtimit të bërë përmes softit. Pikat e kuqe janë nyjet dhe vijat janë skajet. Të gjithë kampionët janë future në soft në përmasa të njëjta, por reduktimi i një pjese të rrjetit ndodh për shkak të lidhshmërisë në vazhdim me pjesë të tjera të qytetit. Në anekset 5 deri në 11, janë gjenerimet e softit për kampionët e Tiranës.

Të dhënat	**Kamp_Tir_1**	**Kamp_Tir_2**	**Kamp_Tir_3**	**Kamp_Tir_4**	**Kamp_Tir_5**	**Kamp_Tir_6**	**Kamp_Tir_7**
Pershkrimi	ind_vern	ind_kom	ind_kom	ind_inf	ind_kom	ind_vern	ind_inf
Sipërfaqja *(km^2)*	0.22	0.20	0.20	0.17	0.20	0.23	0.20
n (numri i nyjeve)	134.00	127.00	123.00	58.00	141.00	132.00	88.00
m (numri i skajeve)	228.00	202.00	225.00	124.00	342.00	219.00	151.00
Gjatësitë totale të rrugëve *(m)*	26,952.53	6,986.80	6,801.89	4,147.52	7,267.10	6,135.54	6,135.54
Gjatësitë totale të skajeve *(m)*	41,879.02	9,536.05	11,036.60	8,295.03	13,771.09	27,098.67	8,690.67
Numri i kryqëzimeve	90.00	92.00	91.00	40.00	114.00	93.00	53.00
Mesatarja e gjatësive së rrugëve *(m)*	42.11	47.53	48.90	66.90	50.37	42.60	56.16
Mesatarja e gjatësive të skajeve *(m)*	41.40	47.21	49.50	66.90	50.26	42.23	57.50
Densiteti i nyjeve për km^2	599.60	632.08	615.74	351.28	695.09	583.00	432.69
Densiteti I kryqëzimeve per km^2	402.77	457.89	455.55	242.25	561.98	410.75	260.59
Densiteti i skajeve për km^2 *(m)*	41,879.03	47,461.70	55,250.37	50,238.65	67,887.58	40,848.34	42,731.84
Densiteti i rrugëve per km^2 *(m)*	26,952.50	34,773.87	34,050.83	25,119.32	35,824.74	27,098.60	25,128.74
Qarkullueshmëria mesatare	1.06	1.10	1.10	1.04	1.11	1.05	1.03
Shkalla mesatare e nyjes	3.40	3.18	3.60	4.27	4.85	3.31	3.40
Mesatarja e rrugëve për nyje	2.30	2.44	2.40	2.30	2.66	2.34	2.22
Densiteti i nyjeve për km^2	599.60	632.08	615.74	351.28	695.09	583.00	432.69
Densiteti i kryqëzimeve për km^2	402.77	457.89	455.55	242.25	561.98	410.75	260.59
Raporti i laqeve të mbyllura	0.006	0.006	<0.001	<0.001	0.010	0.006	<0.001
Qëndërsia i nyjeve	0.46	0.38	0.29	0.47	0.30	0.34	0.39

Tabela 6 Të dhënat e gjeneruara për kampionët e Tiranës. Burimi: autori

4.2.1 Densiteti

Për analogji, në terma të densitetit analiza do të zhvillohet duke krahasuar:

- Mesataren e gjatësive të rrugëve (figura 47)
- Densitetin e kryqëzimeve për km^2 (figura 48)
- Densitetin e rrugëve për km^2 (figura 49)

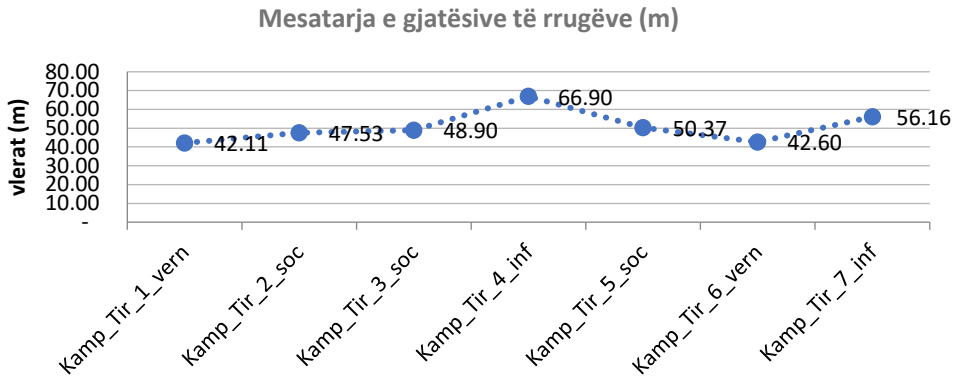


Figure 47 Diagrame - mesatarja e gjatësive të rrugëve, Tiranë. Burimi: autori

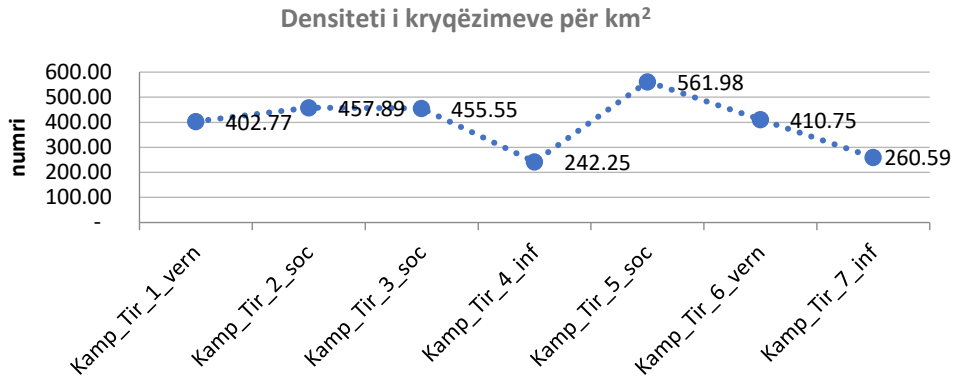


Figure 48 Diagrame - Densiteti i kryqëzimeve për km², Tiranë. Burimi: autori

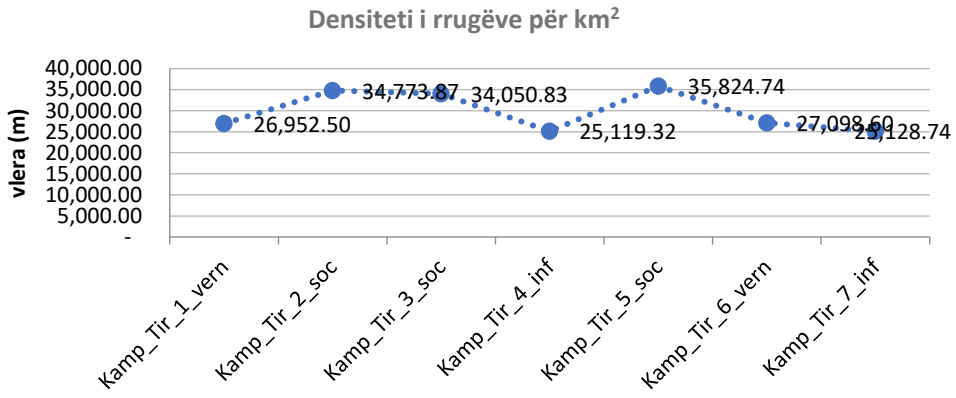


Figure 49 Diagrame - Densiteti i kryqëzimeve për km², Tiranë. Burimi: autori

Pas evidentimit të gjenezës së krijimit të kampionëve, dhe të grupuar si kampion vernakular, kampion i periudhës së socializmit dhe kampion informal, vërehet se kampioni informal ka gjatësi mesatare të rrugës më të lartë dhe dendësi më të ulët. Kampioni vernakular ka gjatësi më të vogël të rrugës dhe dendësi më të lartë. Është interesant fakti që në qytetin e Tiranës, kampioni i periudhës socialiste i përafrohet në vlera atij vernakular. Megjithatë në përfundim, mund të flitet për një rrjet të dendur, me gjatësi mesatare të bllokut të vogël.

4.2.2 Analizë e lidhshmërisë së rrjeteve

Edhe nw nivel kampionwsh, ne do të fokusohemi në parametrat si më poshtë:

- Shkalla mesatare e nyjes (figura 50)
- Mesatarja e rrugëve për nyje (figura 51)
- Qarkullueshmëria mesatare (figura 52)
- Raporti i laqeve të mbyllura (figura 53)

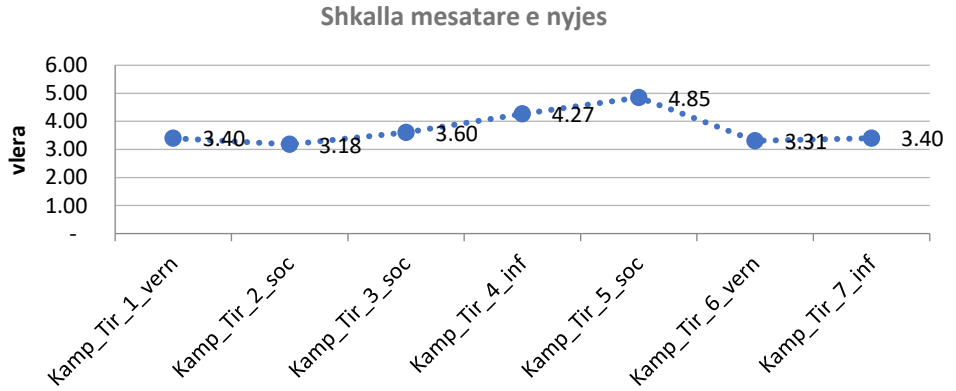


Figure 50 Diagrame – Shkalla mesatare e nyjes, Tiranë. Burimi:autori

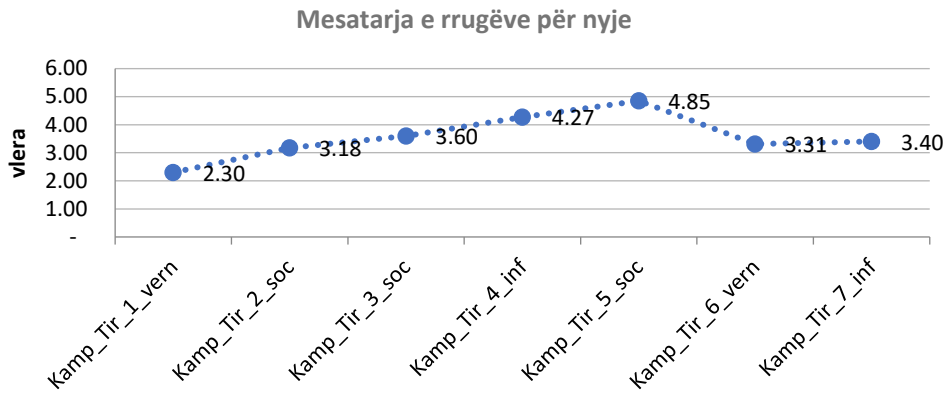


Figure 51 Diagrame – Mesatarja e rrugëve për nyje, Tiranë. Burimi: autori

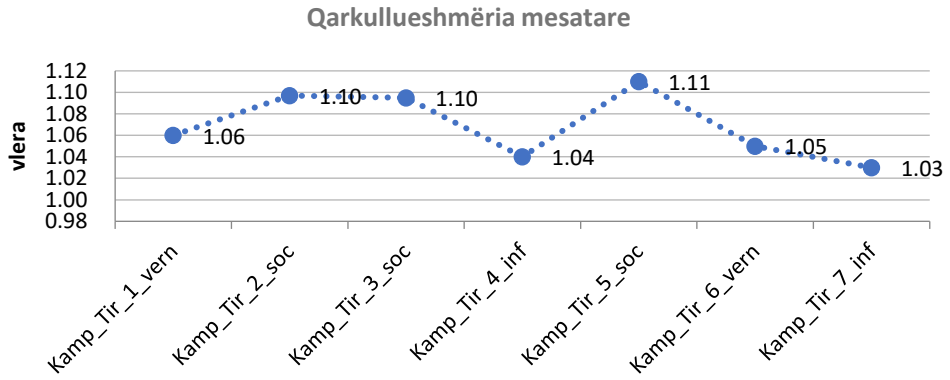


Figure 52 Diagrame – qarkullueshmëria mesatare, Tiranë. Burimi: autori

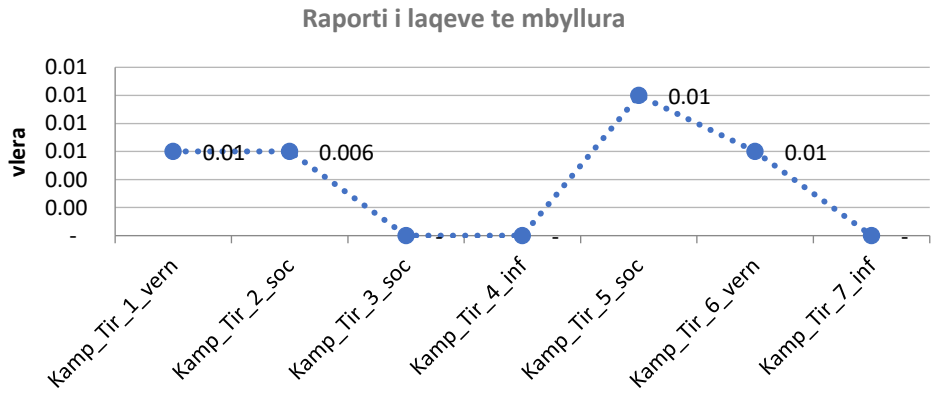


Figure 53 Diagrame – raporti i laqeve të mbyllura, Tiranë. Burimi: autori

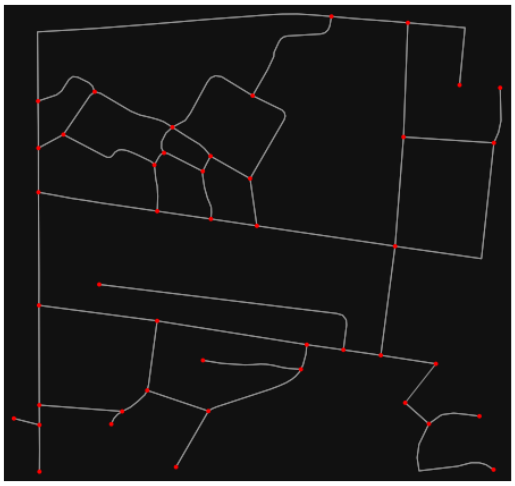
Nisur nga diagramet e mësipërme, ne mund të gjykojmë që në aspektin e lidhshmërisë, kampionët e marre ne studim, paraqiten pothuajse në vlera të qëndrueshme, të përafërta me njëri tjetrin.

Përfundime: **Densifikimi i ndodhur vitet e fundit në Tiranë dhe mbivendosja e kampionëve me gjeneza të ndryshme, artikulohet edhe në aspektin e lidhshmërisë. Qarkullueshmëria paraqitet në vlera që i afrohen shifrës 1, që është dhe idealja. Raporti i rrugëve më laqe të mbyllura, siç vërehet është karakteristikë e kampionit vernakular por deri diku edhe atij socialist në Tiranë**.

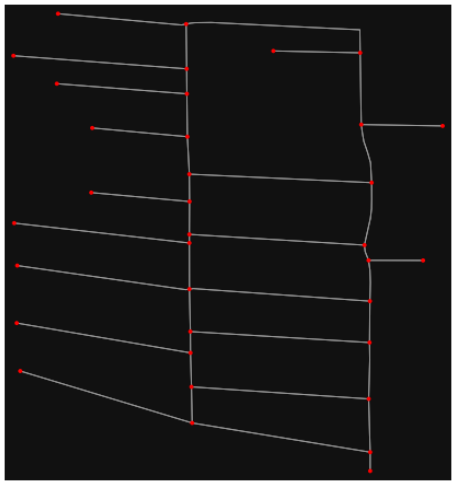
4.3 Rasti i dytë i studimit. Qyteti i Fierit

Figure 54 Përzgjedhja e kampionëve në qytetin e Fierit. Burimi: autori

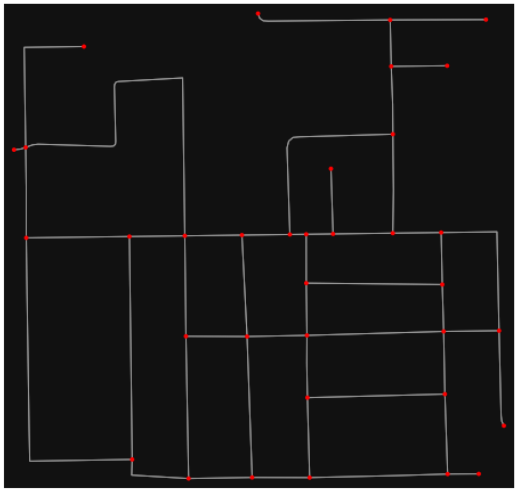
Table 7 Vizualizimi i kampionëve të qytetit Fier përmes OSMnx. Burimi: autori



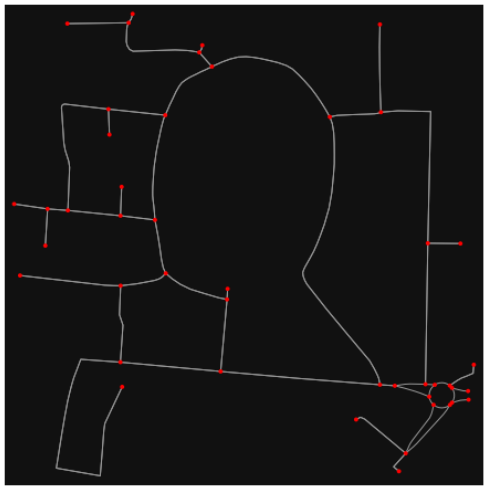
Kamp_Fr_1



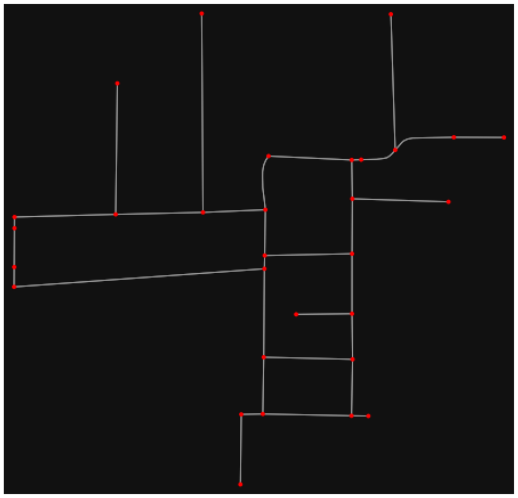
Kamp_Fr_2



Kamp_Fr_3



Kamp_Fr_4



Kamp_Fr_5

Në anekset 12 deri në 16, janë gjenerimet e softit për kampionët e Fierit.

Table 8 Tabela e të dhënave metrike dhe strukturore për qytetin e Fierit. Burimi: autori

Të dhënat	**Kamp_Fr_1**	**Kamp_Fr_2**	**Kamp_Fr_3**	**Kamp_Fr_4**	**Kamp_Fr_5**
Pershkrimi	ind_soc	ind_inf	ind_inf	ind_soc	ind_inf
Sipërfaqja *(km²)*	0.203	0.152	0.201	0.18	0.19
n (numri i nyjeve)	44.00	35.00	35.00	45.00	36.00
m (numri i skajeve)	111.00	80.00	94.00	88.00	110.00
Gjatësitë totale të rrugëve *(m)*	4,612.75	3,618.70	4,601.50	3,372.93	4,078.95
Gjatësitë totale të skajeve *(m)*	9,175.07	7,237.39	9,203.00	6,416.46	8,157.89
Numri i kryqëzimeve	40.00	24.00	32.00	37.00	36.00
Mesatarja e gjatësive së rrugëve *(m)*	82.37	90.47	97.90	66.10	74.16
Mesatarja e gjatësive të skajeve *(m)*	82.65	90.47	97.90	72.90	74.16
Densiteti i nyjeve per km²	216.30	229.45	174.12	251.67	250.72
Densiteti i kryqëzimeve per km²	196.66	157.33	159.20	206.93	192.04
Densiteti i skajeve per km² *(m)*	45,110.49	47,446.18	45,783.84	35,885.79	43,518.94
Densiteti i rrugëve per km² *(m)*	22,679.20	23,723.09	22,891.92	18,864.00	21,759.47
Qarkullueshmëria mesatare	1.08	1.01	1.11	1.16	1.02
Shkalla mesatare e nyjes	5.04	4.57	5.37	3.90	4.68
Mesatarja e rrugëve për nyje	2.75	2.49	3.00	2.46	2.53
Densiteti i nyjeve për km²	216.30	229.45	174.12	251.67	250.72
Densiteti i kryqëzimeve për km²	196.66	157.33	159.20	206.93	192.04
Raporti i laqeve të mbyllura	<0.001	<0.001	<0.002	<0.001	<0.001
Qëndërsia i nyjeve	0.32	0.46	0.33	0.41	0.35

Metodologjia e interpretuar në nënkapitullin e mësipërm, është demonstruar nëpërmjet përdorimit të softit për rastin e qytetit të Fierit.

4.3.1 Densiteti

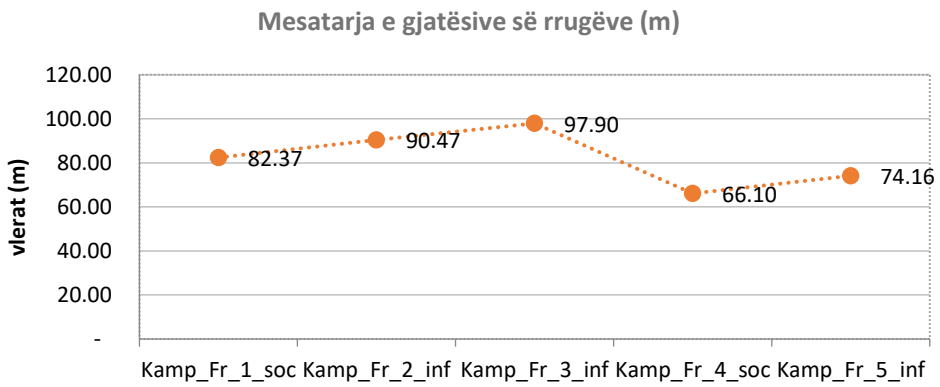


Figure 55 Diagrame – mesatarja e gjatësive të rrugëve, Fier. Burimi: autori

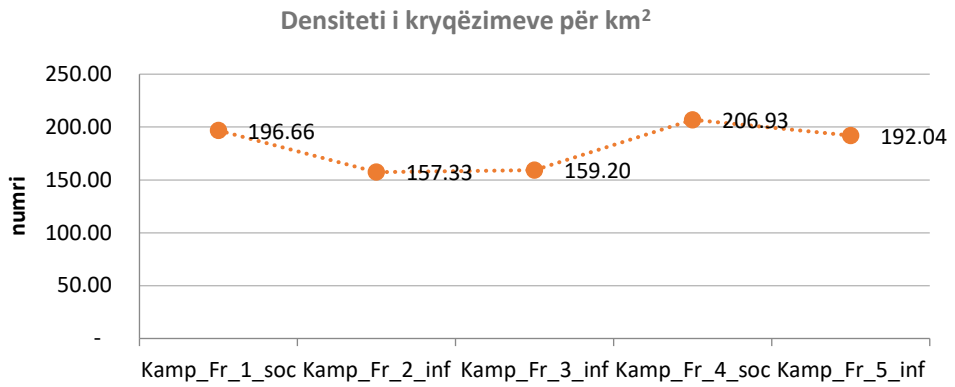


Figure 56 Diagrame – densiteti i kryqëzimeve për km², Fier. Burimi: autori

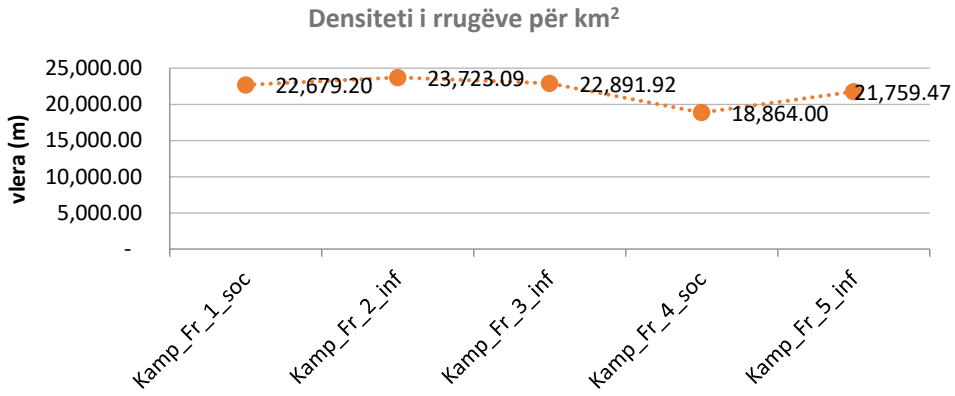


Figure 57 Diagrame – densiteti i rrugëve për km², Fier. Burimi:autori

Në përgjithësi, siç u vërejt edhe në nivel qyteti, Fieri karakterizohet nga rrugë me gjatësi relativisht të madhe me gjatësi nga 65-100 m (figura 55). Më e theksuar kjo mesatare është në kampionët informal. Për rrjedhojë edhe densiteti i kryqëzimeve dhe rrugëve, rezulton më i ulët (figura 56,57). Pra në përfundim të dy tipologjitë e kampionëve reflektojnë rrjet jo të imët dhe densitet të ulët, krahasuar edhe me dy qytetet e tjera.

4.3.2 Analizë e lidhshmërisë së rrjeteve

Është interesant fakti që kampioni informal në Fier shfaq vlera në tilla në aspektin e lidhshmërisë. Duhet të evidentojmë, se zhvillmi informal në Fier ka ndodhur kryesisht në pjesën perëndimore të qytetit, toka ish-bujqësore. Për shkak të zhvillimeve bujqësore para viteve ’90, ato ishin drenazhuar me kanale vaditëse dhe kulluese sipas një rrjeti ortogonal. Gjithashtu, ndarja e parcelave u bë sipas gjeometrisë së krijuar për shfrytëzimin bujqësor. Kjo reflektohet në rrjetin ortogonal të ndërtimit të tyre sot. Për sa i përket qarkullueshmërisë, ky kampion i përafrohet shifrës 1 (figura 58). Ndërsa në aspektin e shkallës mesatare të nyjes qëndron në vlerat 4-4.5 (figura 59). Kampioni i

periudhës socialiste në Fier paraqitet më identitar sesa në Tiranë. Ndërhyrjet kanë qenë të pakta në Fier dhe është ruajtur struktura bazë.

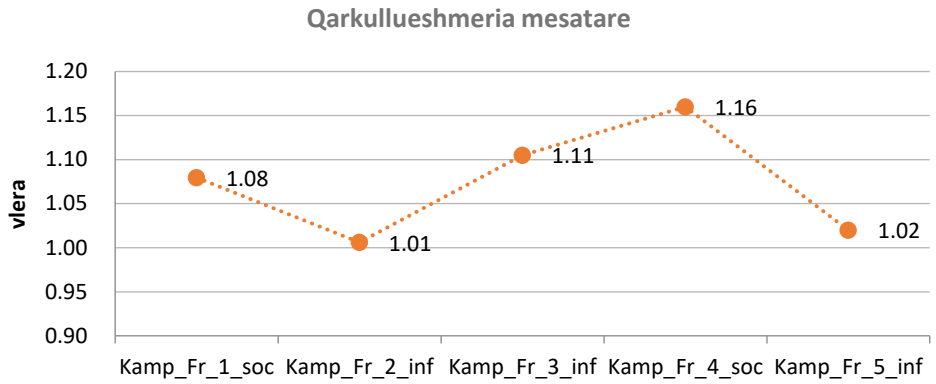


Figure 58 Diagrame – qarkullueshmëria mesatare, Fier. Burimi: autori

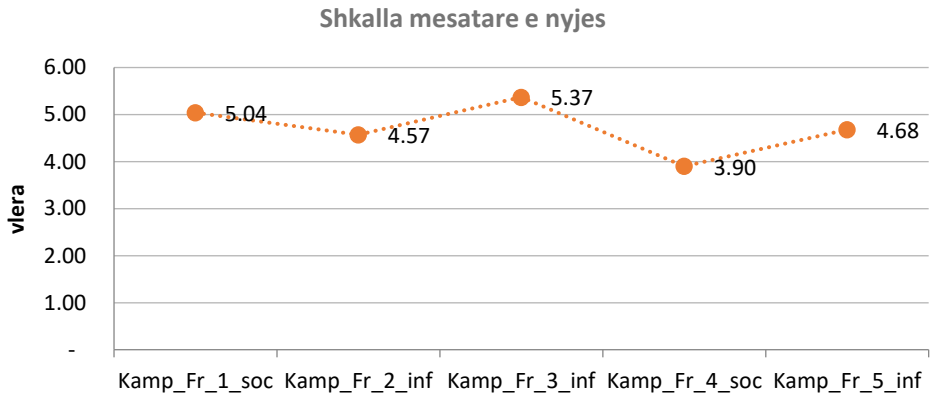


Figure 59 Diagramë – shkalla mesatare e nyjes, Fier. Burimi: autori

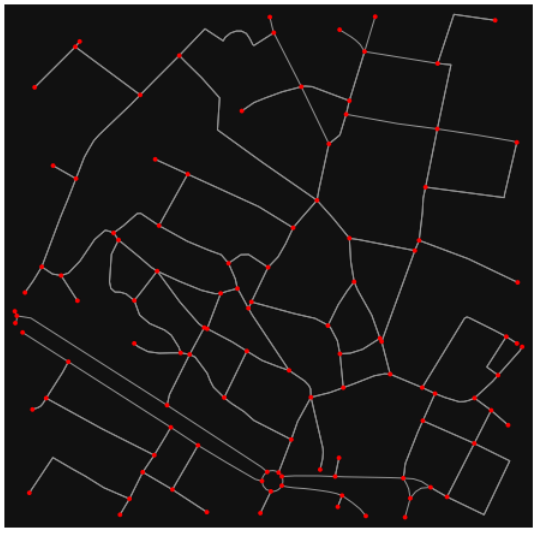
Përfundime:

Kampioni informal, shfaq karakteristika identitare në Fier, rrjet jo i imët, me dendësi të ulët. Nëse krahasojmë të dhënat e këtyre kampionëve me modelin në nivel qyteti, vërejmë që artikulimi i rrjetit të qytetit i përafrohet më shumë periudhës komuniste, kjo lidhet me faktin në atë periudhë është zhvilluar pjesa më e madhe e qytetit.

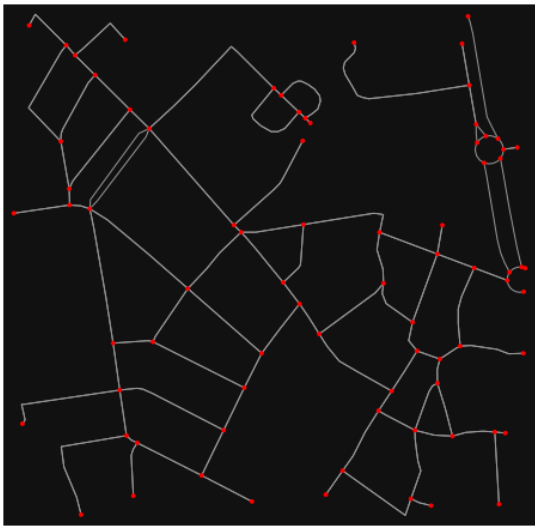
4.4 Rasti i tretë i studimit. Qyteti i Korçës

Figure 60 Përzgjedhja e kampionëve në qytetin e Korçës. Burimi: autori

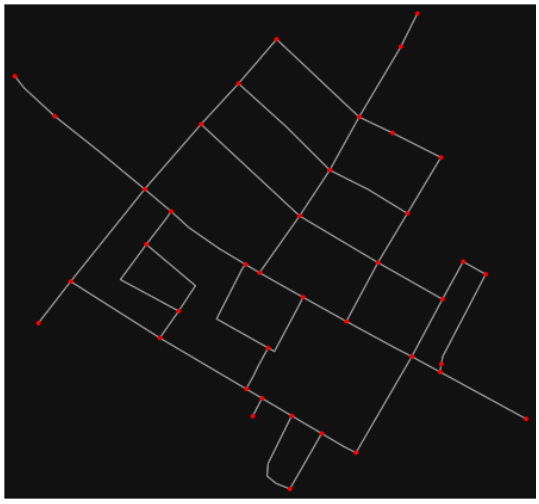
Table 9 Vizualizimi i kampionëve të rrjeteve përmes OSMnx, për Korçën. Burimi: autori



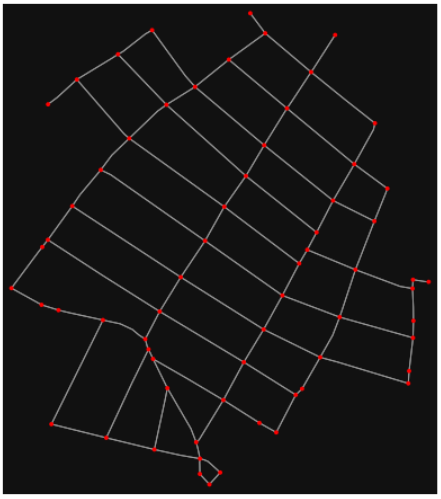
Kamp_Ko_1



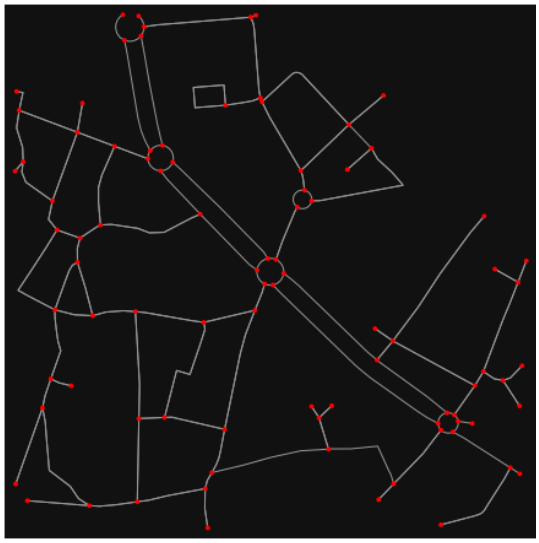
Kamp_Ko_2



Kamp_Ko_3



Kamp_Ko_4



Kamp_Ko_5

Në anekset 17 deri në 21, janë gjenerimet e softit për kampionët e Korçës.

Table 10 Tabela e të dhënave metrike dhe strukturore për qytetin e Korçës. Burimi: autori

Të dhënat	**Kamp_Ko_1**	**Kamp_Ko_2**	**Kamp_Ko_3**	**Kamp_Ko_4**	**Kamp_Ko_5**
Pershkrimi	ind_vern	ind_vern	ind_vern	ind_vern	ind_kom
Sipërfaqja *(km²)*	0.225	0.227	0.216	0.227	0.209
n (numri i nyjeve)	108.00	77.00	101.00	130.00	88.00
m (numri i skajeve)	244.00	181.00	255.00	309.00	182.00
Gjatësitë totale të rrugëve *(m)*	6,815.90	5,493.14	6,998.22	8,116.13	5,422.30
Gjatësitë totale të skajeve *(m)*	12,234.61	10,279.40	13,456.50	13,773.58	9,269.97
Numri i kryqëzimeve	99.00	73.00	96.00	122.00	78.00
Mesatarja e gjatësive së rrugëve *(m)*	**49.75**	**50.48**	**51.08**	**44.80**	**49.29**
Mesatarja e gjatësive të skajeve *(m)*	**50.14**	**51.79**	**52.77**	**44.50**	**50.90**
Densiteti i nyjeve per km²	479.10	339.90	468.45	570.47	420.50
Densiteti I kryqëzimeve per km²	**439.18**	339.90	**445.26**	**535.00**	372.70
Densiteti i skajeve per km² *(m)*	54,275.09	45,378.37	62,413.56	60,442.02	44,298.10
Densiteti i rrugëve per km² *(m)*	**30,236.65**	24,249.43	**32,458.95**	**35,615.70**	25,911.38
Qarkullueshmëria mesatare	1.05	1.08	1.05	1.01	1.06
Shkalla mesatare e nyjes	**4.52**	**4.70**	**5.04**	**4.75**	**4.13**
Mesatarja e rrugëve për nyje	2.81	2.81	2.96	3.06	2.68
Densiteti I nyjeve për km²	479.10	339.90	468.45	570.47	420.50
Densiteti I kryqëzimeve për km²	439.18	339.90	445.26	535.00	372.70
Raporti I laqeve të mbyllura	0.001	0.001	0.001	0.001	0.0090
Qëndërsia i nyjeve	0.26	0.29	0.28	0.22	0.43

4.4.1 Densiteti

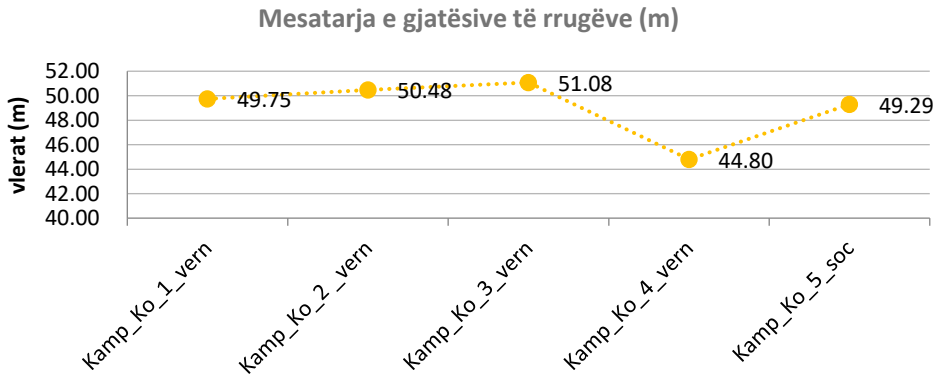


Figure 62 Diagrame - mesatarja e gjatësive të rrugëve, Korçë. Burimi: autori

Nëse i referohemi diagramave të dendësisë rezulton se kampionët në Korçë, reflektojnë gjatësi mesatare të vogël të rrugëve dhe densitet të lartë. Gjatësitë mesatare të bllokut

mesatarisht janë rreth 50 m (figura 62). Kjo lidhet edhe me natyrën kompakte që ka qyteti.

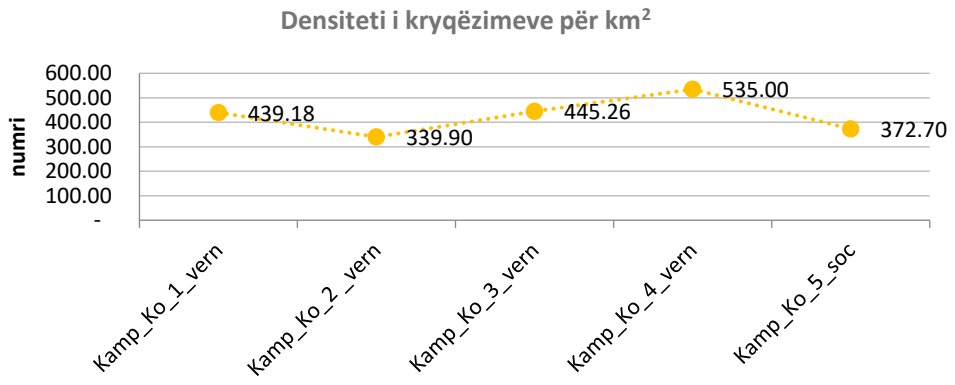


Figure 63 Diagrame – densiteti i kryqëzimeve për km², Korçë. Burimi: autori

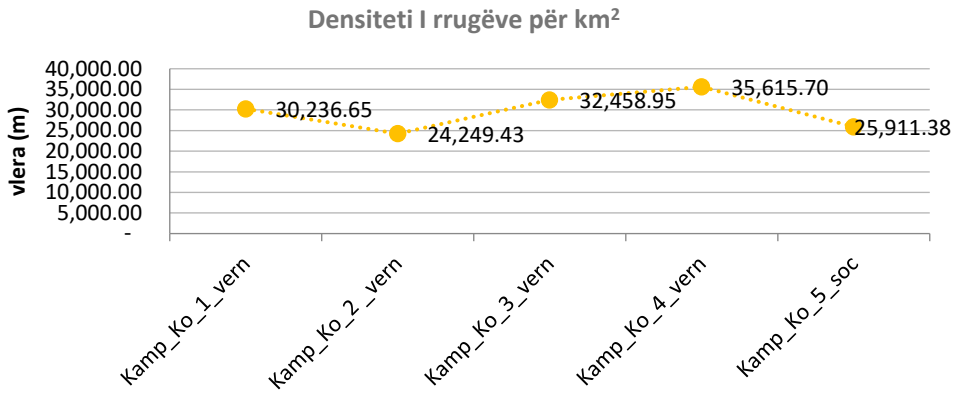


Figure 64 Diagrame – densiteti i rrugëve për km², Korçë. Burimi: autori

4.4.2 Analizë e lidhshmërisë së rrjeteve

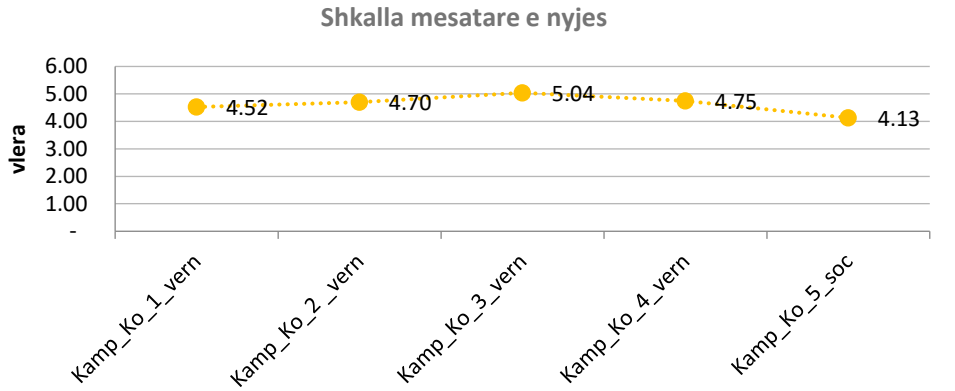


Figure 65 Diagrame – shkalla mesatare e nyjes, Korçë. Burimi: autori

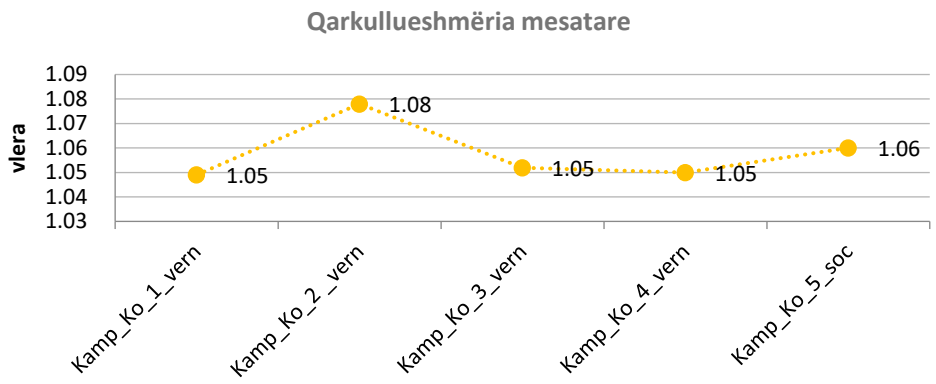


Figure 66 Diagrame – qarkullueshmëria mesatare, Korçë. Burimi: autori

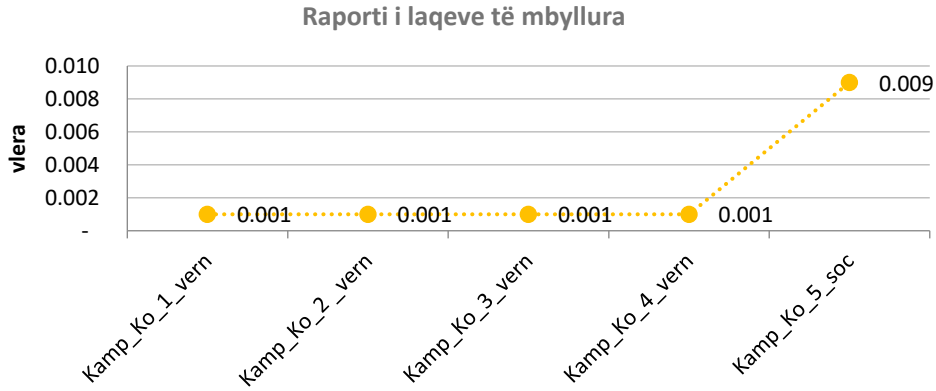


Figure 67 Diagrame – raporti i laqeve të mbyllura, Korçë. Burimi: autori

Lidhshmëria paraqitet e lartë si në aspektin e shkallës mesatare të nyjes (figura 65), ashtu edhe në qarkullueshmërinë mesatare (figura 66) Raporti i laqeve të mbyllura është karakteristikë e kampionëve vernakular, siç vërehet në diagramat e mësipërme (figura 67).

Përfundime: Kampionët e marrë në studim në qytetin e Korçës, në përgjithësi paraqesin vlera të njëjta në aspektin e dendësisë dhe lidhshmërisë. Karakterizohen nga kampionë, me rrjet të imët dhe të dendur. Karakteristikë është raporti i laqeve të mbyllura, të cilat janë karakteristikë e qytetit.

DISKUTIMET E REZULTATEVE 5

5. DISKUTIMET E REZULTATEVE

Pas analizës krahasimore në nivel qytetesh dhe më pas në nivel kampionësh për tu përgjigjur saktë pyetje të kërkimit, lind nevoja e krahasimit të kampionëve urbanë me karakteristika të përbashkëta. Në kapitujt e mëparshëm, nga intuita e gjenezës së tyre, ato janë ndarë në tre grupime të mëdha: (i) vernakular (ii) socialist (iii) informal. Krahasimi i tyre mund të gjenerojë vlera të unfikuara për leximin e strukturës së rrjeteve. Respektivisht, janë marrë në shqyrtim:

(i) Kamp_Tir_1; Kamp_Tir_6; Kamp_Ko_1; Kamp_Ko_3; Kamp_Ko_4;

(ii) Kamp_Tir_2; Kamp_Tir_3; Kamp_Tir_5; Kamp_Fr_1; Kamp_Fr_4;

(iii) Kamp_Tir_4; Kamp_Tir_7; Kamp_Fr_2; Kamp_Fr_3; Kamp_Fr_5

Renditja e tyre, brenda grupimit, është rastësore, për të kuptuar trendin e përgjithshëm dhe arritur në konkluzione.

5.1.1 Densiteti

Në diagramat e mëposhtëm janë krahasuar vlerat e pesë kampionëve në aspektin e densitetit. Nëse I referohemi mesatares së gjatësisë së rrugë, për rrjedhim madhësisë së bllokut, vihet re kampioni vernakular karakterizohet nga gjatësi e qëndrueshme në të gjithë kampionët e marrë dhe konkretisht në vlerat 40-50m; Kampioni I periudhës socialiste ndryshon nga njëri qyte në tjetrin dhe vlerat luhaten nga 50-80 m; ndërsa kampioni informal shfaqet më i çrregullt, në vlera nga 60-100m. Këto vlera tregojnë identitetin e krijuar në kampionët vernakular, ku dendësia është e lartë, rrjeti është më i imët (figura 68).

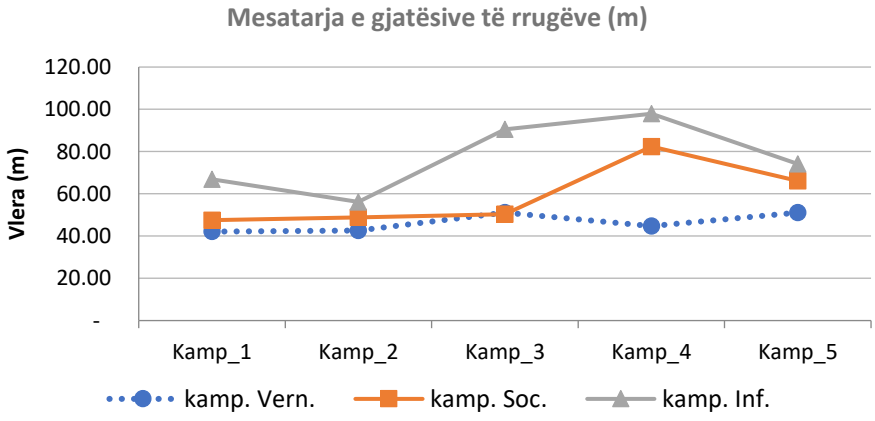


Figure 68 Diagramë krahasuese midis tre kampionëve – mesatarja e gjatësive të rrugëve. Burimi: autori

Për sa i përket densitetit të kryqëzimeve dhe densitetit të rrugëve për km^2, ato nuk mund ti shohim të shkëputura. Siç paraqiten edhe në diagramat e mëposhtme, kampioni vernakular shfaqet në vlera më të qëndrueshme ku mesatarisht janë rreth 400 kryqëzime për km^2 dhe 30 km rrugë për çdo km^2. Kampioni i periudhës socialiste, shfaq vlera të paqëndrueshme nga njëri qytet në tjetrin, megjithatë mesatarisht vihet re densitet i lartë

i kryqëzimeve mbi 400 kryqëzime për km^2 dhe mbi 30 km rrugë për km^2. Kampioni socialist ka synuar të përmirësojë strukturën vernakulare. Për sa i përket kampionit informal, vihet re dendësi më e ulët në rreth 200 kryqëzime për km^2 dhe rreth 25 km rrugë për km^2 (figura 69, 70). Edhe ky kampion shfaqet në vlera të qëndrueshme në qytete të ndryshme.

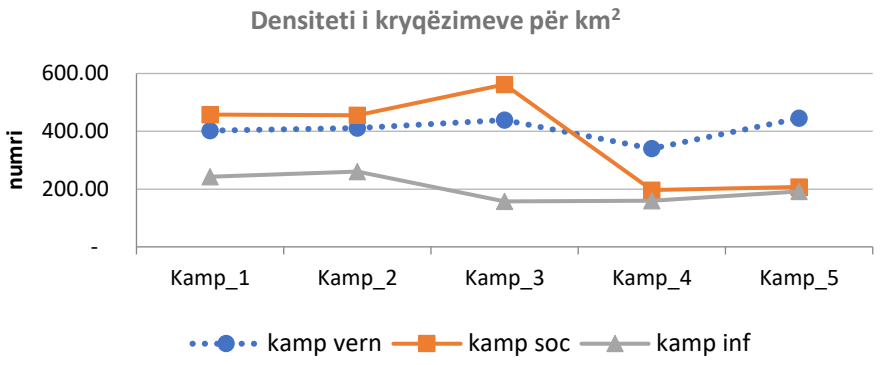


Figure 69 Diagrame krahasuese midis tre kampionëve – densiteti i kryqëzimeve për km^2. Burimi: autori

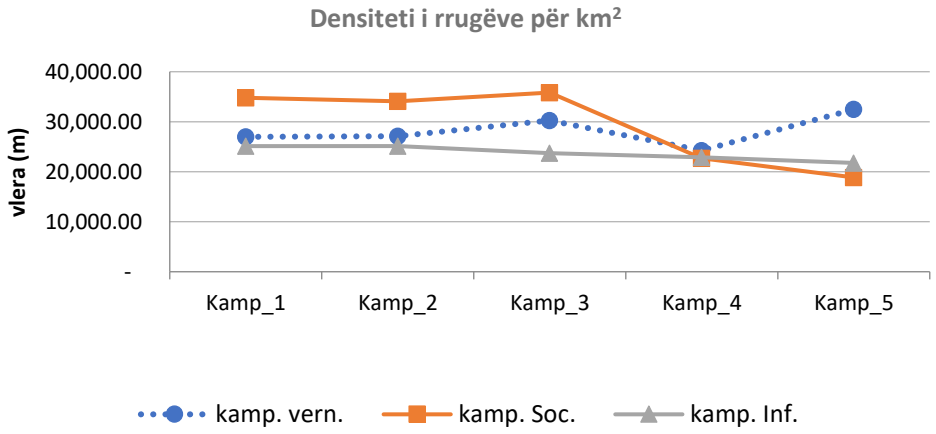


Figure 70 Diagrame krahasuese midis tre kampionëve – densiteti i rrugëve për km^2. Burimi: autori

Në përfundim mund të themi që kampionë me gjeneza të njëjta, në qytete të ndryshme ofrojnë vlera të njëjta në aspektin e dendësisë së rrjeteve, për rrjedhim edhe në leximin e formës urbane.

5.1.2 Analizë e lidhshmërisë së rrjeteve

Në aspektin e lidhshmërisë së rrjeteve, analizohen të njëjtët kampionë në tre komponentë kryesorë: në shkallën mesatare të nyjes, në mesataren e rrugëve për nyje dhe në qarkullueshmërinë mesatare. Dy treguesit e parë, të ndërlidhur me njëri tjetrin, por edhe me treguesit e dendësisë të përmendur më sipër, shfaqen në vlera pothuajse të njëjta. Ka një tendencë të kampionit vernakular të shfaqë vlera pak më të larta për të treguar lidhshmëri më të lartë, por në përfundim mund të themi se qytetet shqiptare

karakterizohen nga shkallë mesatare e nyjes në vlerat nga 3 në 5 dhe mesatarja e rrugëve për nyje nga 2 në 3 (figura 71, 72).

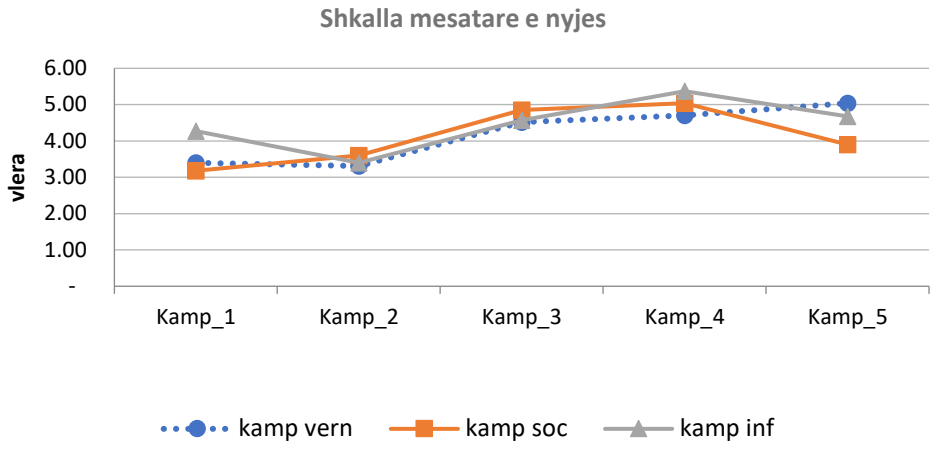


Figure 71 Diagrame krahasuese midis tre kampionëve – shkalla mesatare e nyjes. Burimi: autori

Ndërsa në aspektin e qarkullueshmërisë mesatare, kampioni informal shfaq vlera më afër 1, më pas vjen kampioni vernakular, e më pas ai i periudhës socialiste. Gjithsesi qytet shqiptare shfaqin vlera shumë të larta të qarkullueshmërisë, në raport me qytete të tjera në botë (figura 73).

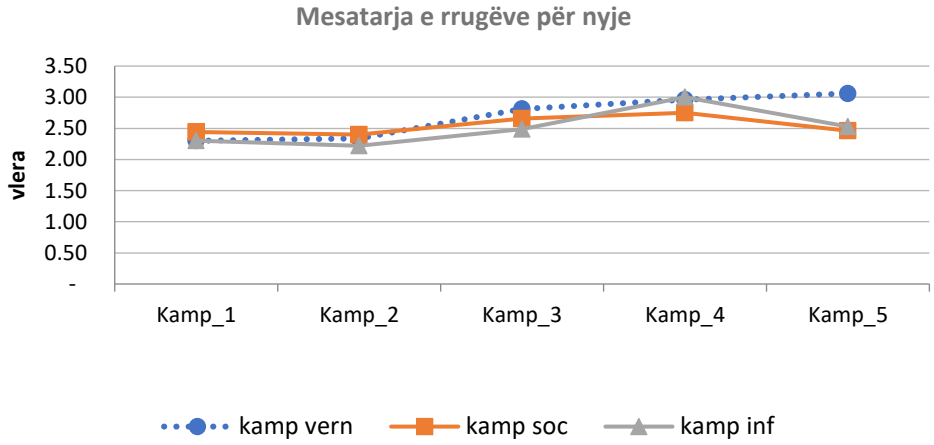


Figure 72 Diagrame krahasuese midis tre kampionëve –mesatarja e rrugëve për nyje. Burimi: autori

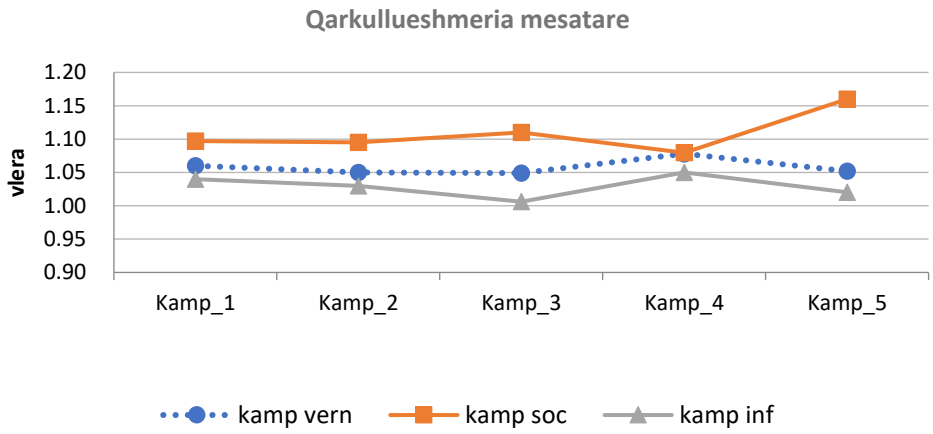


Figure 73 Diagrame krahasuese midis tre kampionëve – densiteti i rrugëve për km². Burimi: autori

5.1.3 Përfundime

Qartësisht është evidente, që leximi i strukturës së rrjeteve ofron një optikë në leximin e formës urbane, çka vërteton dhe hipotezën e parë të ngritur në fillim të këtij studimi. Nëpërmjet matjeve metrike dhe strukturore të rrjetit (densitet, lidhshmëri dhe qendërsi), në nivel qyteti dhe në nivel kampionësh, mund të krijojmë një panoramë më të plotë të natyrës së formës urbane të këtyre qyteteve. Densiteti dhe lidhshmëria ofrojnë kontribut në leximin e formës në të dyja shkallët e observimit, si në nivel qyteti ashtu dhe në nivel kampionësh. Ndërsa qendërsia mund të vëzhgohet vetëm në nivel qyteti. Në aspektin e qendërsisë, mund ti përgjigjemi pyetjes së dytë të kërkimit, që produkti i ndërhyrjeve nga poshtë lart, për zhvillimet adaptive paraqet një strukturë më të qëndrueshme, sesa ato të planifikuara.

Në përfundim mund të themi që kampionë me gjeneza të njëjta, në qytete të ndryshme ofrojnë vlera të njëjta në aspektin e dendësisë dhe lidhshmërisë së rrjeteve. Nëse krahasojmë të dhënat e treguesve të rrjeteve në nivel qyteti, me ato në nivel kampionësh, mund të themi se Korça shfaq veti të indit vernakular, duke demonstruar rrjet të dendur, të imët (me madhësi blloku të vogël), dhe lidhshmëri të lartë. Tirana paraqitet me rrjet të imët dhe të dendur, me lidhshmëri të ulët. Ndërsa Fieri, me rrjet të jo të dendur dhe jo të imët, me lidhshmëri mesatare.

Nëse vërejmë me kujdes kampionët, për ti krahasuar ato më pas në nivel qyteti, konstatohet se kampionët e periudhës komuniste përafrojnë me vlerat e qytetit të Fierit në tërësi, ndërsa Korça më kampionin vernakular. Për sa i përket Tiranës, situata paraqitet më kaotike, duke e lidhur edhe me presionin e madh të zhvillimit të viteve të fundit. Këto përfundime vërtetojnë pjesërisht hipotezën e dytë të kërkimit. Gjithashtu, në bazë të krahasimit të realizuar në nivel kampionësh me gjeneza të ndryshme (vernakulare dhe socialiste), konstatohet që vlerat e tyre mund të unifikohen. Kjo i

përgjigjet edhe pyetjes së parë të kërkimit, që kampionët urbanë të njëjtë, shfaqin të njëjta karakteristika nga një qytet në tjetrin.

Në tre tipologjitë e kampionëve të marrë në studim, indi vernakular shfaqet në vlera më të qëndrueshme si në aspektin e dendësisë ashtu edhe në atë të lidhshmërisë, duke marrë përgjigje kështu dhe pyetja e tretë të kërkimit.

5.1.4 Rekomandime

Nga leximi i formës urbane në nivel qytetesh, rekomandimet e mëposhtme mund të kenë vlera në procesin e planifikimit për qytete e studiuara:

Presioni i zhvillimit në qytetin e Tiranës, ka prishur strukturën e rrjeteve, sidomos në aspektin e lidhshmërisë. Ndërtimet informale dhe problematika e pronave kanë krijuar rrjete të çrregullta, me lidhshmëri të ulët. Struktura rezulton me një densitet të lartë të kryqëzimeve, por me numër të ulët të rrugëve për nyje. Shtimi i numrit të rrugëve në kryqëzimet ekzistuese, do të përmirësonte lidhshmërinë.

Në Fier zona informale shfaq një lidhshmëri shumë të ulët, pavarësisht se është shumë afër qendrës së qytetit. Rekomandohet përmirësimi i lidhshmërisë, si brenda zonës së informale në vetvete, por edhe me pjesën tjetër të qytetit. Mesatarja gjatësive të rrugëve në kampionin informal paraqitet shumë e madhe (90-100 ml). Rekomandohet dendësimi i rrjetit rrugor në këtë zonë.

Në Korçë, ka dendësi dhe lidhshmëri të lartë, rekomandohet që këto vlera të replikohen edhe në pjesën tjetër të qytetit që po ndërtohet rishtazi apo në të ardhmen, për të ruajtur identitetin e saj. Këto aspekte, do e bëjnë atë më atraktiv për turistët, pasi vizitohet lehtësisht në këmbë.

Nga leximi vizual i hartave të qendërsisë të gjeneruara, rekomandohet që rrjetet me qendërsi më të lartë të nyjeve në të tri qytetet të shërbejnë si akset kryesore të lidhshmërisë me unazat, bypass-et apo rrugë nacionale e internacionale. Kjo do të zvogëlonte ngërçet dhe do të përmirësonte qarkullueshmërinë në to.

DISKUTIME 6

6. PËRFUNDIME

6.1.1 Përmirësimet në nivel teorik dhe aplikativ

Në nivel teorik, ky studim përmirëson në dy drejtime:
Së pari, duke vendosur në komunikim disa fusha teorike që konsiderohen të përshtatshme për hapjen dhe adresimin e një kufiri të ri kërkimi, arrin të lexojë formën urbane bazuar në teoritë e rrjeteve. Pas analizimit teorik të formës urbane dhe trajtimit të qyteteve si sisteme komplekse adaptive, u realizuan matje strukturore të kompleksitetit në nivelin e rrjeteve. Teoria e grafeve, është baza ku është zhvilluar softi, i cili shërben për zhvillimin e eksperimentit.

Së dyti, nëpërmjet matjeve strukturore të rrjeteve lexon aftësinë ripërtëritëse që struktura të ndryshme urbane transmetojnë. Vizualizimi i qendërsisë së nyjeve interpreton zhvillimet adaptive në nivel qyteti.

Në nivel aplikativ, ky studim përmirëson:

Të gjithë dakortësojmë që duam qytete më të mira për të jetuar dhe se duam të ruajmë më të mirën e asaj që kemi tashmë brenda mjedisit urban. Ka prova dërrmuese që njerëzit 'vlerësojnë' mjedisin e ndërtuar dhe se ai shton cilësinë e jetës së tyre (nëse janë banorë) dhe përvojat pozitive estetike (nëse janë vizitorë). Për ta bërë këtë, ne duhet të kuptojmë se si u bënë qytetet tona ashtu siç janë - domethënë, cilat procese kanë prodhuar këtë mjedis urban të vlerësuar dhe nga cilat elementë përbëhet në të vërtetë ky mjedis i vlerësuar. Studimi i formës urbane dhe matjet sasiore të elementëve të saj në nivel lokal, na ofrojnë njohuri, kuptim dhe metodologji për të përmirësuar proceset e planifikimit, orientuar vendimmarrësit dhe ruajtjen e identitetit të qyteteve tona.

Kodimi në '*python*' është shumë i lehtë në përdorim dhe nuk kërkon ndonjë ekspertizë të posaçme në fushën e programimit. Me anë të disa rreshtave në programim në një kohë të shkurtër ne mund të nxjerrim rezultate të shpejta. Gjithashtu, vizualizimi i hartave apo gjenerimet që ofron softi, janë të thjeshta dhe lehtësisht të interpretueshme për publikun e gjerë. Ky soft, pa kufizime në përdorim, është në përmirësim të vazhdueshëm nga kërkues nga e gjithë bota dhe lehtësisht mund të përdoret në nivel studimor për kërkime të ndryshme. Gjithashtu mund të ketë përdorim edhe nga institucionet qendrore apo vendore që hartojnë strategji apo inplementojnë projekte në fushën e rrjeteve rrugore.

6.1.2 Pikat e forta të studimit, rekomandime të mëtejshme

Ky studim, u zhvillua vetëm në një drejtim, në analizimin e rrjeteve, duke i konsideruar si *multidigraph*, jo planare, të orientuara. Siç u përmend edhe në fillim, çdo aks që gjeneron lëvizje, u quajt rrjet. Megjithatë, në këtë aspekt, fusha e përdorimit të softit

është shumë e gjerë. Rrjetet mund të përdoren në drejtime të ndryshme, si hapësira kalimi këmbësorësh, biçikletash, rrugë shërbimesh etj. Gjenerimi i tyre mund të hap diskutime të tjera në aspektin e transportit apo lëvizjen e këmbësorëve, apo dhe hierarkizimin e tyre. Nëpërmjet tij, mund të zhvillohen matje të kompleksitetit të rrjetit lëvizjet këmbësore, rrugët e biçikletave apo me makinë. Ka një mori fushash kërkimore, ku lloji i lëvizjes është matës për zhvillimet ekonomike, sociale dhe mjedisore.

Ky studim zhvillon një dimension të ri, në matjen e rrjeteve për sa i përket shkallës së tyre (në shkallë qyteti), falë përdorimit të softit *OSMnx*. Duke qenë se krijimi i bazës së të dhënave, realizohet nëpërmjet *OpenStreetMap*, azhornimi i tij ngelet sfidë. Me qëllim zhvillimin e matjeve jo vetëm në nivel rrjetesh, por edhe masës së ndërtuar apo qëllime të tjera, ngelet detyrë azhornimi i tyre në terren, ose plotësimi nëpërmjet platformave të tjera të informacionit qoftë shtetërore ose jo. Kjo vlen më tepër në hapësirat ku mungon baza e të dhënave.

Gjithashtu OpenStreetMap ofron bazën e të dhënave për pikat e interesit në një qytet. Nëpërmjet OSMnx, mund të gjenerohen studime të ndryshme, lidhur me arritshmërinë e tyre në kohë dhe zhvillimin e kërkimeve në aspektin e zhvillimeve turistike në një zonë ose rajon.

Së fundi, mjete dhe teknikat e thjeshta të vizualizimit mund t'i ndihmojnë planifikuesit të përcjellin krahasimin e formës urbane tek njerëzit e zakonshëm. Ata mund të kuptojnë se si qytetet evoluojnë, si ndryshon densiteti dhe lidhshmëria apo të krahasojnë qytete të ndryshme me njëri tjetrin.

Së bashku, teknologjitë e informacionit hapësinor dhe morfologjia urbane do të konvergojnë më tej për të gjeneruar kuptime të reja të së shkuarës dhe të tashmes së qytetit dhe për të fuqizuar planifikuesit dhe anëtarët e komunitetit në proceset e vendimmarrjes bashkëpunuese të drejtuara nga të dhënat që përqendrojnë përvojën njerëzore fiziologjike, psikologjike dhe sociale.

7. Lista e figurave:

8. Lista e tabelave:

9. Bibliografia:

Alexander, C. (1965, April). A city is not a Tree. *Architectural Forum, 122*(1), 58-62.

Aligica, P. D. (2014). *Institutional Diversity and Political Economy: The Ostroms and Beyond.* Oxford: Oxford University Press. doi:10.1093/acprof:oso/9780199843909.001.0001

Aziz-Alaoui, M., & Bertelle, C. (2012). *From system complexity to emergent properties.* Berlin: Springer Berlin Heidelberg.

Bak, P., Tang, C., & Wiesenfeld, K. (1987). Self-organized criticality: An explanation of the 1/f noise. *Physical Review Letters*, 381-384.

Banerjee, T. (2002). *City sense and city design: Writings and projects of Kevin Lynch.* MIT Press.

Barrington-Leigh, C., & Millard-Ball, A. (2015). A century of sprawl in the United States. *ENVIRONMENTAL SCIENCES*.

Barthélemy, M. (2011). Spatial networks. *Physics Reports*, 1-101. doi:https://doi.org/10.1016/j.physrep.2010.11.002

Barthélemy, M., & Flammini, A. (2008). Modeling Urban Street Patterns. *Physical Review Letters*.

Batty, M. (2005). *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals.* Cambrige: MIT Press.

Batty, M. (2009). Urban modeling. *International Encyclopedia of human Geography*, 51-58.

Batty, M. (2013 b). Resilient Cities, Networks, and Disruption. *Environment and Planning B: Planning and Design*, 571-573.

Batty, M. (2013). *The New Science of Cities.* Cambridge, MA: MIT Press.

Batty, M., & Longley, P. A. (1994). *Fractal cities: a geometry of form and function.* Academic press.

Batty, M., & Marshall, S. (2011). The Origins of Complexity Theory in Cities and Planning. Në J. Portugali, H. Meyer, E. Stolk, & E. Tan, *Complexity Theories of Cities Have Come of Age* (fv. 21-45). Springer Science + Business Media.

Batty, M., & Xie, Y. (1999). Self-organized criticality and urban development. *Discrete Dynamics in Nature and Society*, 109-124. doi:https://doi.org/10.1155/S1026022699000151

Baynes, T. M. (2009). Complexity in Urban Development and Management. *Journal of Industrial Ecology*, 214-227. doi: https://doi.org/10.1111/j.1530-9290.2009.00123.x

Beineke, L. W., Oellermann, O. R., & Pippert, R. E. (2002). The average connectivity of a graph. *Discrete Mathematics, 252*(1), 31-45.

Benjamin, W. (1937). *Paris, the capital of the nineteenth century.* President and Fellows of Harvard College (2002).

Boeing, G. (2017). OSMnx: A Python package to work with graph-theoretic OpenStreetMap street networks. *Journal of Open Source Sofware*.

Boeing, G. (2017a). OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. *Computers, Environment and Urban Systems*, 126-139. doi:doi:10.1016/j.compenvurbsys.2017.05.004

Boeing, G. (2017c). OSMnx: New Methods for Acquiring, Constructing, Analyzing, and. *Computers, Environment and Urban Systems, 65*, 126-139. doi:10.1016/j.compenvurbsys.2017.05.004

Boeing, G. (2019). Urban spatial order: street network orientation, configuration, and entropy. *Applied Network Science.*

Boeing, G. (2020). Exploring Urban Form Through Openstreetmap Data: A Visual Introduction. Në J. Hollander, & A. Sussman, *Urban Experience and Design: Contemporary Perspectives on Improving the Public Realm* (fv. 167-184). Routledge.

Boeing, G. (2020). Off the Grid…and Back Again? *Journal of the American Planning Association*, 123-137.

Boeing, G. D. (2017). Methods and Measures for Analyzing Complex Street Networks and Urban Form. USA, California: University of California, Berkeley.

Boeing, G., & Waddell, P. (2016). New Insights into Rental Housing Markets across the United States: Web Scraping and Analyzing Craigslist Rental Listings. *Journal of Planning Education and Research*. doi:https://doi.org/10.1177/0739456X16664789

Bonchev, D., & Buck, G. A. (200). Quantitative Measures of Network Complexity. Në D. Bonchev, & D. H. Rouvray, *Complexity in Chemistry, Biology, and Ecology* (fv. 191-235). Boston, MA: Springer.

Calvino, I., & Weaver, W. (1974). *Le città invisibli [Invisible Cities]* . London: Secker & Warburg.

Caniggia, G., & Maffei, G. L. (2003). *Gianfranco Caniggia: architetto Roma (1933-1987) : disegni, progetti, opere.* Alinea Editrice.

Caniggia, G., Maffei, G. L., & Galán, M. G. (1995). *Tipología de la edificación: estructura del espacio antrópico* (bot. i 1. ed.(12/1995)). Celeste Ediciones.

Castells, M. (2009). *The Rise of the Network Society, 2nd Edition, with a New Preface.* Malden,MA: Wiley-Blackwell.

Cavalcante, A., Mansouri, A., Kacha, L., & Barros, A. K. (2014). Measuring streetscape complexity based on the statistics of local contrast and spatial frequency. *PLUS ONE*.

Cervero, R., & Kockelman, K. (1997). Travel demand and the 3DS: Density, diversity, and design. *Transportation Research Part D: Transport and Environment*, 199-219.

Cilliers, P. (1998). *Complexity and Postmodernism: Understanding Complex Systems.* London: Routledge.

Comunian, R. (2011). Rethinking the Creative City: The Role of Complexity, Networks and Interactions in the Urban Creative Economy. *Urban Studies, 48*, 1157-1179. doi:10.1177/0042098010370626

Conzen, M. (1960). *Alnwick, Northumberland: a study in town-plan analysis.* London: George Philip: Institute of British Geographers. doi:https://doi.org/10.1177/0309132509334948

Conzen, M. G. (2004). *hinking about urban form: Papers on urban morphology, 1932-1998.* Peter Lang.

Corcoran, P., Mooney, P., & Bertolotto, M. (2013). Analysing the growth of OpenStreetMap networks. *Spatial Statistics*, 21-32. doi:https://doi.org/10.1016/j.spasta.2013.01.002

Dankelmann, P., & Oellermann, O. R. (2003). Bounds on the average connectivity of a graph. *Discrete Applied Mathematics, 129*(2-3), 305-318. doi:https://doi.org/10.1016/S0166-218X(02)00572-3

Devlin, K. (2000). *The Language of Mathematics: Making the Invisible Visible.* New York: W.H. Freeman.

Dill, J. (2004). Measuring Network Connectivity for Bicycling and Walking. *Presented at the 83rd Annual Meeting of the Transportation Research Board.* Washington DC.

Downey, A. B. (2012). *Think Complexity: Complexity Science and Computional Modeling.* Sebastopo, CA: Reilly Media.

Elezaj, L., & Fleury, M. (2015). *Can you read Tirana?* Ecole Polytechnique Fédérale de Lausanne.

Ewing, R., & Clemente, O. (2013). *Measuring Urban Design: Metrics for Livable Places.* Washington : Island Press.

Fecht, S. (2012, April 6). Grid Unlocked: How Street Networks Evolve as Cities Grow. *Scientific American*.

Fetch, S. (2012, April 6). Grid Unlocked: How Street Networks Evolve as Cities Grow. *Scientific Americal*.

Fortier, B. (1989). *La métropole imaginaire; Un atlas de Paris.* Institut français d'architecture.

Frankhauser, P. (2008). Fractal geometry for measuring and modelling urban patterns. Në S. Albeverio, D. Andrey, P. Giordano, & A. Vancheri, *The Dynamics of Complex Urban Systems* (fv. 241-243). Physica Heidelberg.

Frankhauser, P. (2008). Fractal geometry for measuring and modelling urban patterns. Në S. Albeverio, D. G. Andrey, & A. Vancheri, *The Dynamics of Complex Urban Systems* (fv. 241-243). Springer.

Freeman, L. C. (1977). A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 35-41. doi:https://doi.org/10.2307/3033543

Gastner, M. T., & Newman, M. E. (2006). The spatial structure of networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 247-252.

Gershenson, C., & Fernández, N. (2012). Complexity and Information: Measuring Emergence, Self-organization, and Homeostasis at Multiple Scales. *Complexity*, 29-44. doi:10.1002/cplx.21424

Giacomin, D. J., & Levinson, D. M. (2015). Road network circuity in metropolitan areas. *Environment and Planning B: Planning and Design*, 1040-1053. doi:https://doi.org/10.1068/b130131p

Glaeser, E. (2011). *Triumph of the City: How Our Greatest Invention Makes Us Richer, Smarter, Greener, Healthier, and Happier.* New York: Penguin Press.

Goodchild, M. F. (2007). Citizens as sensors: the world of volunteered geography. *GeoJournal*, *69*(4), 211–221. doi:10.1007/s10708-007-9111-y

Gudmundsson, A., & Mohajeri, N. (2013). Entropy and order in urban street networks. *Scientific Reports, 3*.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, (fv. 47-57). New York. doi:https://doi.org/10.1145/602259.602266

Haken, H. (2012). Complexity and Complexity Theories: Do These Concepts Make Sense? Në J. Portugali, H. Meyer, E. Stolk, & E. Tan, *Complexity Theories of Cities Have Come of Age* (fv. 7-20). Springer Complexity.

Haklay, M. (2010). How Good is Volunteered Geographical Information? A Comparative Study of OpenStreetMap and Ordnance Survey Datasets. *Environment and Planning B: Urban Analytics and City Science*, 682–703. doi:https://doi.org/10.1068/b35097

Hall, P. (1996). *Design, Cities of Tomorrow: An Intellectual History of Urban Planning and Design Since.* Malden, MA: Blackwell Publishers.

Heinzle, F., Anders, K.-H., & Sester, M. (2006). Pattern Recognition in Road Networks on the Example of Circular Road Detection. *Geographic Information Science.* GIScience.

Hiller, B. (1996). *Space is the machine: A configurational theory of architecture.* Cambridge: Cambridge University Press.

Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity.* Helix Books.

Holling, C. S. (1973). Resilience and Stability of Ecological Systems. *Annual Review of Ecology, Evolution, and Systematics*, 1-23. doi:http://dx.doi.org/10.1146/annurev.es.04.110173.000245

Howard, E. (2003). *To-morrow: A peaceful path to real reform.* Routledge .

http://www.urbanform.org/. (2016).

J, W., & Conzen, M. (1981). *The urban landscape.* Academic Press.

Jacobs, A. B. (1995). *Great Streets.* Cambridge: The MIT Press.

Jacobs, J. (1961). *Death and life of great american cities.* New York: Vintage Books (1992 edition).

Jacobs, J. (1969). *The economy of Cities.* New York: NY:Vintage Books.

Jenks, M., Williams, K., & Burton, E. (2000). *Achieving sustainable urban form.* London: E & FN Spon. doi:DOI:10.1016/S0264-8377(01)00010-2

Jenks, M., Williams, K., & Burton, E. (2000). *Achieving Sustainable Urban Form.* E&FN Spoon. doi:10.1016/S0264-8377(01)00010-2

Kansky, K. J. (1963). *Structure of transportation networks : relationships between network geometry and regional characteristics.* Chicago: Univ. of Chicago / Dep. of Geography.

Kansky, K. J. (a.d.). *Structure of transportation networks : relationships between network geometry and regional characteristics.*

Karmici. (1888). *Gjeografia e Korçës dhe e rrethit.* Selanik: AIH.

Kitchin, R. (2016). The ethics of smart cities and urban science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciencs*, 374. doi:https://doi.org/10.1098/rsta.2016.0115

Kitchin, R. (2017). Urban Science: A Short Primer (Working Paper No.23). *Project: The Programmable City*. Maynooth: National University of Ireland, Maynooth.

Knight, P. L., & Marshall, W. E. (2015). The metrics of street network connectivity: their inconsistencies. *Journal of Urbanism: International Research on Placemaking and Urban Sustainability*, 241-259. doi:https://doi.org/10.1080/17549175.2014.909515

Krönert, R., Volk, M., & Steinhardt, U. (2001). Introduction: Landscape balance and landscape assessment. Në R. Krönert, M. Volk, & U. Steinhardt, *Landscape Balance and Landscape Assessment* (fv. 1-22).

Kuper, R. (2017). Evaluations of landscape preference, complexity, and coherence for designed digital landscape models. *Landscape and Urban Planning, 157*, 407-421. doi:https://doi.org/10.1016/j.landurbplan.2016.09.002

Lance, N. (2004). Urban Forms: The Death and Life of The Urban Block by Philippe Panerai, Jean Castex, Charles DePaule, Ivor Samuels. *Landscape Journal, 24*, 209-211. doi:10.3368/lj.24.2.209

Levinson, D., & El-Geneidy, A. M. (2009). The minimum circuity frontier and the journey to work. Unpublished. *Regional Science and Urban Economics 39(6):732-738, 39*(6), 732-738. doi:10.1016/j.regsciurbeco.2009.07.003

Levi-Strauss, C. (1955). *Tristes tropiques (Terre humaine).* Paris.

Lloyd, S. (2001). *Measures of complexity: a nonexhaustive list* (Vëll. i 21). IEEE Control Systems Magazine.

Lynch, K. (1954). The form of Cities. *Scientific American*, 54-63.

Lynch, K. (1960). *The Image of the City.* The MIT Press.

Mandelbrot, B. B. (1983). *The fractal geometry of nature.* New York: W.H. Freeman&Co.

MarcBarthélemy. (2011). Spatial networks. *Physics Reports*, 1-101. doi:https://doi.org/10.1016/j.physrep.2010.11.002

Maron, M. (2015, November 19). *How complete is OpenStreetMap?* Gjetur në Mapbox: https://blog.mapbox.com/how-complete-is-openstreetmap-7c369787af6e

Marshall, S. (2005). *Streets and Patterns.* New York,: NY: Spon Press.

Marshall, S. (2009). *Cities Design and Evolution* (bot. i Ilustrated). Routledge.

Marshall, S. (2012 , November 07). Science, pseudo-science and urban design. *Springerlink*, 257–271. doi:https://doi.org/10.1057/udi.2012.22

Marshall, S. (2012). Planning, Design and the Complexity of Cities. Në J. Portugali, H. Meyer, E. Stolk, & E. Tan, *Complexity Theories of Cities Have Come of Age* (fv. 191-205). Berlin: Springer-Verlag.

Marshall, W. E., Piatkowski, D. P., & Garrick, N. W. (2014). Community Design, street networks, and Public Health. *Journal of Transport & Health*, 326-340.

Marshall, W., & Garrick, N. (2010). Street network types and road safety: A study of 24 California cities. *URBAN DESIGN International*, 133-147.

Massuci, A. P., D, S., A, C., & M, B. (2009, September). Random planar graphs and the London street network. *The European Physical Journal B: Condensed Matter and Complex Systems, 71(2)*, 259-271. doi:doi:10.1140/epjb/e2009-00290-4

Masucci, A. P., D, S., Crooks, A., & Batty, M. (2009). Random planar graphs and the London street network. *The European Physical Journal B: Condensed Matter and Complex Systems*, 259-271. doi:10.1140/epjb/e2009-00290-4

Masucci, A. P., Smith, D., Crooks, A., & Batty, M. (2009). Random planar graphs and the London street network. *The European Physical Journal B: Condensed Matter and Complex Systems, 71*(2), 259-271. doi:10.1140/epjb/e2009-00290-4

Mëhilli, S. (2014). *Tirana (1920-1944).* Tiranë: Mediaprint.

Miller, J. H., & Page, S. E. (2007). *Complex adaptive systems: An introduction to computational models of social life.* Princeton, Oxford: Princeton University Press.

Mitchell, M. (2009). *Complexity: A Guided Tour.* New York: Oxford University Press.

Mohajeri, N., & Gudmundsson, A. (2014). The evolution and complexity of Urban Street Networks. *Geographical Analysis*, 345-367.

Mohajeri, N., French, J. R., & Batty, M. (2013). Evolution and entropy in the Organization of Urban Street Patterns. *Annals of GIS*, 1-16.

Mohajeri, N., French, J., & Gudmundsson, A. (2013). Entropy Measures of Street-Network Dispersion: Analysis of Coastal Cities in Brazil and Britain. *Entropy*, 3340-3360.

Mohajeri, N., Gudmundsson, A., & French, J. R. (2015). CO2 emissions in relation to street-network configuration and city size. *Transportation Research Part D: Transport and Environment*, 116-129.

Moudon, A. (1997). Urban morphology as an emerging interdisciplinary field. *Urban morphology*, 3-10.

Moudon, A. (1997). Urban morphology as an emerging interdisciplinary field. Seattle: Urban Morphology.

Mudon, A. (1994). Getting to know the built landscape: typomorphology. Në K. A. Franck, & L. H. Schneekloth, *Ordering space : types in architecture and design* (fv. 289-311). New York: Van Nostrand Reinhold.

Muratori, S. (1959). *Studi per una operante storia urbana di Venezia.* Rome: Instituto poligrafico dello Stato, Libreria dello Stato.

Newell, G. F. (1980). *Traffic flow on transportation networks.* Cambridge: MIT Press.

Newman, M. (2010). *Networks: An Introduction.* Oxford: Oxford University Press.

O'Sullivan, D. (2013). Spatial network analysis. *Handbook of Regional Science*, 1253-1273.

Oliveira, V. (2016). *Urban Morphology: An Introduction to the Study of the Physical Form of Cities.* Springer.

Oliveira, V. (2019). Urban Forms, Agents, and Processes of Change. Në L. D'Acci, *The Mathematics of Urban Morphology.* Springer International Publishing.

Over, M., Schilling, A., & Neubauer, S. (2010). Generating web-based 3D City Models from OpenStreetMap: The current situation in Germany. *Computers Environment and Urban Systems, 34(*(6), 496-507. doi:10.1016/j.compenvurbsys.2010.05.001

P.Pepo. (1972). Materiale dokumentare për ambientin shoqëror të Varoshit të Korçës. *Studime historike*.

Panerai, P., Samuels, I., Castex, J., & Depaule, C. J. (2004). *Urban Forms. The Death and Life of the Urban Block.* Architectural Press.

Parrott, L. (2010). Measuring ecological complexity. *Ecological Indicators, 10,* 1069-1076.

Pojani, D., & Boussauw, K. (2014). Keep the children walking: Active School travel in Tirana, Albania. *Journal of Transport Geography*, 55-65.

Porta, P., Vito, C., & Latora, V. (2006). The network analysis of urban streets: A dual approach. *Physica A: Statistical Mechanics and its Applications*, 853-866.

Portugali, J. (2000). *Self-Organization and the City.* Springer-Verlag Berlin Heidelberg.

Portugali, J. (2011). *Complexity Theories of Cities: Achievements, Criticism and Potentials.* Springer Science + Business Media.

Portugali, Y., & Stolk, E. (2016). What Makes Cities Complex? *Complexity, Cognition, Urban Planning and Design: Post-Proceedings of the 2nd Delft International Conference* (fv. 3-19). Switzeland: Springer.

PPV Fier. (2016, 12). *https://planifikimi.gov.al/.* Gjetur në https://planifikimi.gov.al/index.php?id=761

Proulx, R., & Parrott, L. (2008). Measures of structural complexity in digital images for monitoring the ecological signature of an old-growth forest ecosystem. *Ecological Indicators*, 270-284. doi:10.1016/j.ecolind.2007.02.005

Rossi, A. (1986). *The Architecture of the City.* Cambridge: MIT Press.

Roy, A. (2005). Urban Informality: Toward an Epistemology of Planning. *Journal of the American Planning Association*, 147-159. doi:https://doi.org/10.1080/01944360508976689

Salat, S., Bourdic, L., & Nowacki, C. (2010). Assessing urban complexity. *International Journal of Sustainable Building Technology and Urban Development*, 160-167.

Salat, S., Bourdic, L., & Nowacki, C. (2010). Assessing Urban Complexity. *International Journal of Sustainable Building Technology and Urban Development*, 160-167. doi:https://doi.org/10.5390/SUSB.2010.1.2.160

Salat, S., Labbé, F., Nowacki, C., & Walker, G. (2011). *Cities and forms : on sustainable urbanism.* Paris : CSTB Urban Morphology Laboratory.

Salingaros, N. (1998). Theory of the urban web. *Journal of Urban Design*, 53-71. doi:10.1080/13574809808724416

Salingaros, N. (2005). *Principles of Urban Structure.* Amsterdam: Techne Press.

Salingaros, N. A. (2000). Complexity and Urban Coherence. *Journal of Urban Design*, 291-316. doi:https://doi.org/10.1080/713683969

Scott, J. C. (2020). *Seeing like a state: How certain schemes to improve the human condition have failed.* New Haven ; London: Yale University Press.

Sevtsuk, A. (2014). Location and agglomeration. *Journal of Planning Education and Research*, 374-393.

Sevtsuk, A., & Mekonnen, M. (2012). Urban network analysis. A new toolbox for ArcGIS. *Revue Internationale de Géomatique, 22*(2), 287-305.

Sevtsuk, A., Kalvo, R., & Ekmekci, O. (2016). Pedestrian accessibility in grid layouts: the role of block, plot and street dimensions. *Urban Morphology, 20*(2), 89-106.

Shanken, A. M. (2018). The Visual Culture of Planning. *Journal of Planning History*, 300-319. doi:https://doi.org/10.1177/1538513218775122

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal, 27*, 379-423, 623-656.

Shannon, C. E. (2001). A mathematical Theory of Communication. *Association for Computing Machinery*, 3-55.

Shiner, J. S., Davison, M., & Landsberg, P. T. (1999). Simple measure for complexity. *Physical Review E, 59*, 1459. doi:https://doi.org/10.1103/PhysRevE.59.1459

Shpuza, E. (2021). Transformation of urban form in Shkodër during. *Journal of Design for Resilience in Architecture and Planning, 2*(Special issue), 115-128.

Silva, P. (2016). Tactical urbanism: Towards an evolutionary cities' approach? *Environment and Planning B: Urban Analytics and City Science*, 1040-1051. doi:https://doi.org/10.1177/0265813516657340

Smith, M. E. (2007). Form and Meaning in the Earliest Cities: A New Approach to Ancient Urban Planning. *Journal of Planning History*, 3-47. doi:https://doi.org/10.1177/1538513206293713

Southworth, M., & Ben-Joseph, E. (1997). *Streets and the Shaping of Towns and Cities.* New York: McGraw-Hill.

Southworth, M., & Ben-Joseph, E. (1997). *Streets and the Shaping of Towns and Cities.* New York: McGraw-Hill.

Speranza, P. (2016). Using parametric methods to understand place in urban design courses. *Journal of Urban Design, 21*(5), 661-689. doi:https://doi.org/10.1080/13574809.2015.1092378

Strano, E., Nicosia, V., Latora, V., Porta, S., & Barthelemy, M. (2012). Elementary processes governing the evolution of road networks. *Scientific Reports*.

Strano, E., Viana, M., Costa, L. d., Cardillo, A., Porta, S., & Latora, V. (2013). Urban Street Networks, a Comparative Analysis of Ten European Cities. *Environment and Planning B: Urban Analytics and City Science*, 1071-1086. doi:doi:10.1068/b38216

Strano, E., Viana, M., Costa, L. d., Cardillo, A., Porta, S., & Latora, V. (2013). Urban Street Networks, a Comparative Analysis of Ten European Cities. *Evironment and Planning B: Planning and Design* , 1071-1086.

Strappa, G. (2018). Reading the Built Environment as a Design Method. Në V. Oliveira, *Teaching Urban Morphology.* Springer International Publishing.

Summers, L., & Johnson, S. (2016). Does the Configuration of the Street Network Influence Where Outdoor Serious Violence Takes Place? Using Space Syntax to Test Crime Pattern Theory. *Journal of Quantitative Criminology*, 397-420.

Sussman, A., & Hollander, J. B. (2015). *Cognitive Architecture: Designing for How We Respond to the Built Environment.* New York: Routledge.

Talen, E. (2003). Measuring Urbanism: Issues in Smart Growth Research. *Journal of Urban Design, 8*(3), 195-215. doi:https://doi.org/10.1080/1357480032000155141

Talen, E. (2011). *City Rules: How Regulations Affect Urban Form.* Washington: Island Press.

Thomo, P. (2022). *Korça, urbanistika dhe arkitektura.* Tirana: Berk.

Tinkler, K. J. (1979). Graph theory. *Progress in Human Geography*, 85-116. doi:https://doi.org/10.1177/030913257900300104

TR030. (2016). *www.tirana.al.* Gjetur në https://tirana.al/faqe/plani-i-pergjithshem-vendor

Urban, D., & Keitt, T. (2001). Landscape Connectivity: A Graph-Theoretic Perspective. *Ecology, 82*(5), 1205-1218.

urbanism, C. f. (2016). *www.cnu.org*.

Usui, H., & Asami, Y. (2011). An evaluation of road network patterns based on the criteria for fire-fighting. *Cybergeo*.

Veizaj, D. (2016). Indeksi I fragmentit dhe dendësia e rrjetit të lëvizjes si mjete analitike të gjurmës fizike të indit urban. *Indeksi I fragmentit dhe dendësia e rrjetit të lëvizjes si mjete analitike të gjurmës fizike të indit urban*. Tirana: U.P.T, Fakulteti I Arkitekturës dhe Urbanistikës. Departamenti I Arkitekturës.

Viana, M. P., Strano, E., Bordin, P., & Barthelemy, M. (2013). The simplicity of planar networks. *Scientific Reports*, 1-6. doi:10.1038/srep03495

Visscher, P. M., Yang, J., & Goddard, M. E. (2010). A Commentary on 'Common SNPs Explain a Large Proportion of the Heritability for Human Height' by Yang et al. *Twin Research and Human Genetics*, 517-524.

Walker, B., Holling, C., & Carpenter, S. R. (2004). Resilience, Adaptability and Transformability in Social-Ecological Systems. *Ecology and Society*. doi:10.5751/ES-00650-090205

Whitehand, J. (2007). Conzenian Urban Morphology and Urban Lanscapes. *Proceedings of the 6th International Space Syntax Symposium* (fv. 01-09). Instanbul: Instanbul: Technical University Faculty of Architecture.

Whitehand, J. W. (2011). British urban morphology: The Conzenian tradition. *Urban Morphology*, 103-109.

Zook, M., Graham, M., Shelton, T., & Gorman, S. (2010). Volunteered Geographic Information and Crowdsourcing Disaster Relief: A Case Study of the Haitian Earthquake. *World Medical & Health Policy*, *2*(2), 7-33. doi:https://doi.org/10.2202/1948-4682.1069

10. Anekse:

ANEKSI 1: QYTETI TIRANË

Once you've perused the features demo notebook, this notebook demonstrates more details on querying for place boundaries and street networks, visualizing, and saving models to disk.

In [1]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  pd.Int64Index,
```

In [5]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf([ "W176158039", "W762985073","W176446171", "W176156312"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.54921 40.72994, 19.55218 40.72903)	40.729944	40.729034	19.552184	19.549206	143246605	way	176158039	40.729944	19.549206	Rruga Shaban Serdani, Afrim i Ri, Fier, Bashki...	highway	residential	0.1
1	LINESTRING (19.55964 40.72787, 19.55996	40.727868	40.727669	19.560292	19.559636	331392960	way	762985073	40.727761	19.559965	Bulevardi Jakov Xoxa, 29 Nëntori, Fier,	highway	secondary	0.1

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
	40.727...										Bashki...			
2	LINESTRING (19.55956 40.74040, 19.55719 40.740...	40.747639	40.740173	19.559555	19.541649	142747315	way	176446171	40.740257	19.547852	Rruga Aleksandra Mano, Qendër, Bashkia Fier, F...	highway	tertiary	0.1
3	LINESTRING (19.57015 40.73416, 19.57014 40.734...	40.737284	40.734161	19.570150	19.567295	142773648	way	176156312	40.735589	19.569405	Rruga Vangjel Maci, Bishanakë, Sheqi Vogël, F...	highway	residential	0.1

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Tirana
north, south, east, west = 40.74, 40.72, 19.56, 19.54

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [7]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [8]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
```

```
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [9]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- Part 5: calculate basic network indicators¶

In [10]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[10]:

2.5365166655489237

In [11]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[11]:

10727

In [12]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[12]:

68353503.77726677

In [13]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[13]:

{'n': 14911,
 'm': 33779,
 'k_avg': 4.530749111394273,
 'edge_length_total': 1774651.6060000046,
 'edge_length_avg': 52.53712679475427,
 'streets_per_node_avg': 2.5365166655489237,
 'streets_per_node_counts': {0: 0,
  1: 4184,
  2: 46,
  3: 9205,
  4: 1450,
  5: 25,
  6: 1},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.28059821608208707,
  2: 0.0030849708269063106,
  3: 0.6173294882972302,
  4: 0.0972436456307424,
  5: 0.0016766145798403863,
  6: 6.706458319361545e-05},
 'intersection_count': 10727,
 'street_length_total': 988113.2849999953,
 'street_segment_count': 18767,
 'street_length_avg': 52.65163771513802,
 'circuity_avg': 1.0675637446906574,
 'self_loop_proportion': 0.002717536100602121,
 'clean_intersection_count': 4895,
 'node_density_km': 218.14536455348687,
 'intersection_density_km': 156.9341644132019,
 'edge_density_km': 25962.847665904494,
 'street_density_km': 14455.927354065283,
 'clean_intersection_density_km': 71.61300781230757}

In [14]:

```
import pandas as pd
```

In [15]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[15]:

	value
n	14911
m	33779
k_avg	4.530749
edge_length_total	1774651.606

	value
edge_length_avg	52.537127
streets_per_node_avg	2.536517
streets_per_node_counts	{0: 0, 1: 4184, 2: 46, 3: 9205, 4: 1450, 5: 25...
streets_per_node_proportions	{0: 0.0, 1: 0.28059821608208707, 2: 0.00308497...
intersection_count	10727
street_length_total	988113.285
street_segment_count	18767
street_length_avg	52.651638
circuity_avg	1.068118
self_loop_proportion	0.002718
node_density_km	218.145365
intersection_density_km	156.934164
edge_density_km	25962.847666
street_density_km	14455.927354
0way_int_count	0
1way_int_count	4184
2way_int_count	46
3way_int_count	9205
4way_int_count	1450
5way_int_count	25
6way_int_count	1
0way_int_prop	0.0
1way_int_prop	0.280598
2way_int_prop	0.003085
3way_int_prop	0.617329
4way_int_prop	0.097244
5way_int_prop	0.001677
6way_int_prop	0.000067

In [ ]:

In [16]:

```
import networkx as nx
```

In [17]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [18]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=1, node_size=0)
```

In [19]:

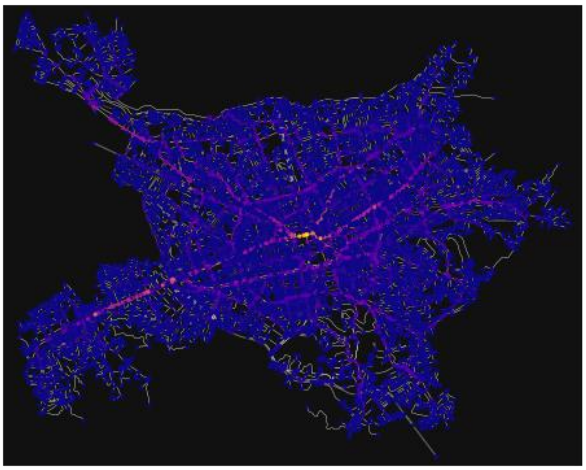
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[19]:

(6274653625, 0.14816688266518646)

In [20]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=10,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 2: QYTETI FIER

In [41]:

```
import geopandas as gpd
```

```
import osmnx as ox

%matplotlib inline
ox.__version__
```

Out[41]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [42]:

```
                                        # turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [43]:

```
# get the boundary polygon for Fier, project it, and plot it
city = ox.geocode_to_gdf("Fier, Fier, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [44]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
   ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  pd.Int64Index,

In [45]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf([ "W176158039", "W762985073","W176446171", "W176156312"], by_osmid=True)
```

Out[45]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.54921 40.72994, 19.55218 40.72903)	40.729944	40.729034	19.552184	19.549206	143246605	way	176158039	40.729944	19.549206	Rruga Shaban Serdani, Afrim i Ri, Fier, Bashki...	highway	residential	0.1
1	LINESTRING (19.55964 40.72787, 19.55996 40.727...	40.727768	40.727669	19.560292	19.559636	331392960	way	762985073	40.727761	19.559965	Bulevardi Jakov Xoxa, 29 Nëntori, Fier, Bashki...	highway	secondary	0.1
2	LINESTRING (19.55956 40.74040, 19.55	40.747639	40.740173	19.559555	19.541649	142747315	way	176446171	40.740257	19.547852	Rruga Aleksandra Mano, Qend	highway	tertiary	0.1

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
	719 40.740...										ër, Bashkia Fier, F...			
3	LINESTRING (19.57015 40.73416, 19.57014 40.734...	40.74372 84	40.74134 161	19.570150	19.567295	142773648	way	176156312	40.735589	19.569405	Rruga Vangjel Maci, Bishanakë, Sheqi Vogël, F...	highway	residential	0.1

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [46]:
```
# define a bounding box in Fieri
north, south, east, west = 40.74, 40.72, 19.56, 19.54

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [47]:
```
                    # create the street network within the city of Fieri's borders
G = ox.graph_from_place("Fier, Fier, Albania", network_type="drive")
```

In [48]:
```
                      # you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Fier, Fier, Albania",
    {"city": "Fier", "state": "Albania"},
    "Fier, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [49]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- Part 5: calculate basic network indicators¶

In [50]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[50]:

2.6533333333333333

In [51]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[51]:

1101

In [52]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[52]:

9718946.585816322

In [53]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[53]:

{'n': 1425,
 'm': 3536,
 'k_avg': 4.9628070175438594,
 'edge_length_total': 280675.7199999999,
 'edge_length_avg': 79.3766176470588,
 'streets_per_node_avg': 2.6533333333333333,
 'streets_per_node_counts': {0: 0, 1: 324, 2: 0, 3: 951, 4: 147, 5: 2, 6: 1},
 'streets_per_node_proportions': {0: 0.0,

```
 1: 0.22736842105263158,
 2: 0.0,
 3: 0.6673684210526316,
 4: 0.1031578947368421,
 5: 0.0014035087719298245,
 6: 0.0007017543859649122},
'intersection_count': 1101,
'street_length_total': 145281.58899999992,
'street_segment_count': 1858,
'street_length_avg': 78.1924590958019,
'circuity_avg': 1.053051386424685,
'self_loop_proportion': 0.0016146393972012918,
'clean_intersection_count': 785,
'node_density_km': 146.6208284424181,
'intersection_density_km': 113.2838821860367,
'edge_density_km': 28879.232694787483,
'street_density_km': 14948.285569551497,
'clean_intersection_density_km': 80.77007040512154}
```

In [54]:

```
import pandas as pd
```

In [55]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[55]:

	value
n	1425
m	3536
k_avg	4.962807
edge_length_total	280675.72
edge_length_avg	79.376618
streets_per_node_avg	2.653333
streets_per_node_counts	{0: 0, 1: 324, 2: 0, 3: 951, 4: 147, 5: 2, 6: 1}
streets_per_node_proportions	{0: 0.0, 1: 0.22736842105263158, 2: 0.0, 3: 0....
intersection_count	1101
street_length_total	145281.589
street_segment_count	1858
street_length_avg	78.192459
circuity_avg	1.053366
self_loop_proportion	0.001615
node_density_km	146.620828

	value
intersection_density_km	113.283882
edge_density_km	28879.232695
street_density_km	14948.28557
0way_int_count	0
1way_int_count	324
2way_int_count	0
3way_int_count	951
4way_int_count	147
5way_int_count	2
6way_int_count	1
0way_int_prop	0.0
1way_int_prop	0.227368
2way_int_prop	0.0
3way_int_prop	0.667368
4way_int_prop	0.103158
5way_int_prop	0.001404
6way_int_prop	0.000702

In [ ]:

In [56]:
```
import networkx as nx
```

In [57]:
```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [58]:
```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [59]:
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[59]:
(7344581268, 0.1932191445513909)

In [60]:
```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```

ANEKSI 3: QYTETI KORCË

In [1]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

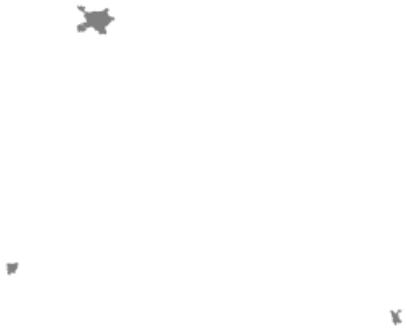
- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Korçë, Korçë County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```



In [5]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf([ "W176158039", "W303328908","W176446171", "W176156312"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.54921 40.72994, 19.55218 40.72903)	40.729944	40.729034	19.552184	19.549206	143246605	way	176158039	40.729944	19.549206	Rruga Shaban Serdani, Afrim i Ri, Fier, Bashki...	highway	residential	0.1
1	LINESTRING (20.78237 40.61794, 20.78244 40.6188...	40.624199	40.617945	20.782965	20.782374	174277070	way	303328908	40.621146	20.782960	Bulevardi Republika, Korçë, Bashkia Korçë, Kor...	highway	tertiary	0.1
2	LINESTRING (19.55956 40.74040, 19.55719	40.747639	40.740173	19.559555	19.541649	142747315	way	176446171	40.740257	19.547852	Rruga Aleksandra Mano, Qendër, Bash	highway	tertiary	0.1

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
	40.740...										kia Fier, F...			
3	LINESTRING (19.57015 40.73416, 19.57014 40.734...	40.737284	40.734161	19.570150	19.567295	142773648	way	176156312	40.735589	19.569405	Rruga Vangjel Maci, Bishanakë, Sheqi Vogël, F...	highway	residential	0.1

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Korca
north, south, east, west = 40.6528, 40.5859, 20.8945, 20.7200

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [7]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Korçë, Korçë County, Albania", network_type="drive")
```

In [8]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Korçë, Korçë County, Albania",
    {"city": "Korçë", "state": "Albania"},
    "Korçë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [9]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- Part 5: calculate basic network indicators¶

In [10]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[10]:
2.899276236429433

In [11]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[11]:

1431

In [12]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[12]:

12093251.026099099

In [13]:
```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[13]:

{'n': 1658,
 'm': 4352,
 'k_avg': 5.249698431845597,
 'edge_length_total': 261743.1450000007,
 'edge_length_avg': 60.14318589154428,
 'streets_per_node_avg': 2.899276236429433,

'streets_per_node_counts': {0: 0, 1: 227, 2: 3, 3: 1143, 4: 281, 5: 3, 6: 1},
'streets_per_node_proportions': {0: 0.0,
 1: 0.13691194209891436,
 2: 0.0018094089264173703,
 3: 0.6893848009650181,
 4: 0.16948130277442702,
 5: 0.0018094089264173703,
 6: 0.0006031363088057901},
'intersection_count': 1431,
'street_length_total': 139246.28400000033,
'street_segment_count': 2378,
'street_length_avg': 58.55604878048795,
'circuity_avg': 1.0425274716948594,
'self_loop_proportion': 0.00042052144659377626,
'clean_intersection_count': 795,
'node_density_km': 137.10126387203744,
'intersection_density_km': 118.33046357110106,
'edge_density_km': 21643.737026141163,
'street_density_km': 11514.379689918402,
'clean_intersection_density_km': 65.73914642838949}

In [14]:
```
import pandas as pd
```

In [15]:
```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[15]:

	value
n	1658
m	4352
k_avg	5.249698
edge_length_total	261743.145
edge_length_avg	60.143186
streets_per_node_avg	2.899276
streets_per_node_counts	{0: 0, 1: 227, 2: 3, 3: 1143, 4: 281, 5: 3, 6: 1}
streets_per_node_proportions	{0: 0.0, 1: 0.13691194209891436, 2: 0.00180940...
intersection_count	1431
street_length_total	139246.284
street_segment_count	2378
street_length_avg	58.556049
circuity_avg	1.042629
self_loop_proportion	0.000421

	value
node_density_km	137.101264
intersection_density_km	118.330464
edge_density_km	21643.737026
street_density_km	11514.37969
0way_int_count	0
1way_int_count	227
2way_int_count	3
3way_int_count	1143
4way_int_count	281
5way_int_count	3
6way_int_count	1
0way_int_prop	0.0
1way_int_prop	0.136912
2way_int_prop	0.001809
3way_int_prop	0.689385
4way_int_prop	0.169481
5way_int_prop	0.001809
6way_int_prop	0.000603

In [ ]:

In [16]:
```
import networkx as nx
```

In [17]:
```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [18]:
```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=1, node_size=0)
```

In [19]:
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
```

```
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[19]:

```
(4477282829, 0.2331623444966312)
```

In [20]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=10,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```

ANEKSI 4: ORIENTIMI I RRUGËVE

In [16]:

```
import matplotlib.pyplot as plt
import numpy as np
import osmnx as ox

%matplotlib inline
weight_by_length = False

ox.__version__
```

Out[16]:

```
'1.2.0'
```

In [17]:

```
# define the study sites as label : query
places = {
   "Fieri": "Fier, Fier, Albania",
   "Korçë": "Korçë, Korçë County, Albania",
   "Tiranë": "Tiranë, Tirana County, Albania"
}
```

In [18]:

```
# verify OSMnx geocodes each query to what you expect (i.e., a [multi]polygon geometry)
gdf = ox.geocode_to_gdf(list(places.values()))
gdf
```

Out[18]:

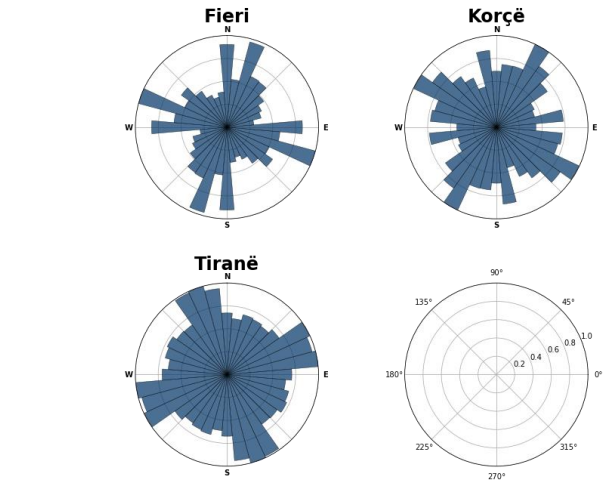
	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	POLYGON ((19.54282 40.72483, 19.54894 40.72301...	40.74186	40.71831	19.579939	19.542816	282231998	relation	1251483	40.726826	19.560450	Fier, Bashkia Fier, Fier County, Southern Alba...	boundary	administrative	0.807769
1	POLYGON ((20.76156 40.63221, 20.76169 40.63119...	40.63701	40.59872	20.797705	20.761564	282415197	relation	1252581	40.617888	20.778459	Korçë, Bashkia Korçë, Korçë County, Southern A...	boundary	administrative	0.944613
2	POLYGON ((19.75473 41.31208, 19.75636	41.36543 6	41.29553 5	19.876438	19.754734	282281489	relation	1250106	41.330514	19.825563	Tiranë, Tirana Municipality, Tirana Coun	boundary	administrative	1.033763

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
	41.3113 6...										ty, Ce...			

In [20]:

```
# create figure and axes
n = len(places)
ncols = int(np.ceil(np.sqrt(n)))
nrows = int(np.ceil(n / ncols))
figsize = (ncols * 5, nrows * 5)
fig, axes = plt.subplots(nrows, ncols, figsize=figsize, subplot_kw={"projection": "polar"})

# plot each city's polar histogram
for ax, place in zip(axes.flat, sorted(places.keys())):
    print(ox.utils.ts(), place)

    # get undirected graphs with edge bearing attributes
    G = ox.graph_from_place(place, network_type="drive")
    Gu = ox.add_edge_bearings(ox.get_undirected(G))
    fig, ax = ox.bearing.plot_orientation(Gu, ax=ax, title=place, area=True)

# add figure title and save image
suptitle_font = {
    "family": "DejaVu Sans",
    "fontsize": 60,
    "fontweight": "normal",
    "y": 1,
}
fig.suptitle("City Street Network Orientation", **suptitle_font)
fig.tight_layout()
fig.subplots_adjust(hspace=0.35)

fig.savefig("images/street-orientations.png", facecolor="w", dpi=100, bbox_inches="tight")
plt.show()
```

2022-06-21 18:01:29 Fieri
2022-06-21 18:01:32 Korçë
2022-06-21 18:01:36 Tiranë



You can also calculate the orientation entropy of a spatial graph with the ox.bearing.orientation_entropy function.
In [ ]:

In [ ]:

In [ ]:

In [ ]:

ANEKSI 5: KAMPIONI URBAN TIRANA 1

In [34]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

Out[34]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [35]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

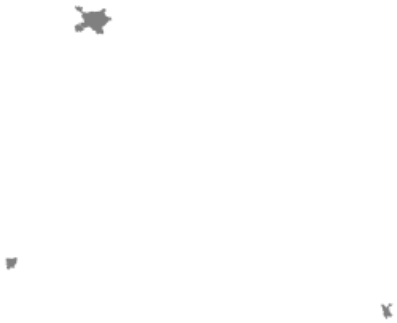
OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [36]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [37]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  pd.Int64Index,
```



In [38]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W174503324"], by_osmid=True)
```

Out[38]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.82761 41.33	41.33575	41.32969	19.827609	19.82624	142609722	way	174503324	41.333453	19.82733	Rruga Dervish Hekali,	highway	residential	0.1

geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
358, 19.82733 41.333...										Selvia, Njësia Bashkiake ...			

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box*¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [39]:
```
# define a bounding box in Tirana 1st Zone
north, south, east, west = 41.3360, 41.3310, 19.8240, 19.8290

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

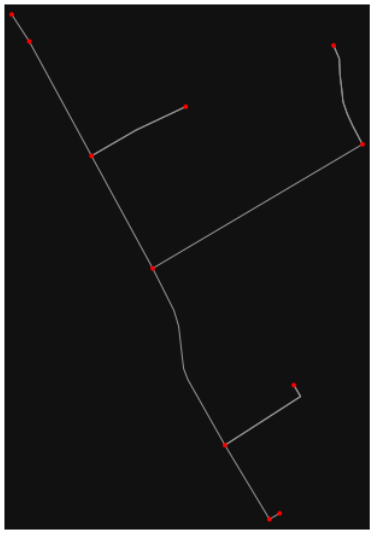
In [40]:
```
# define a point at the corner of the interested zone
location_point = (41.33323, 19.82686)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [41]:

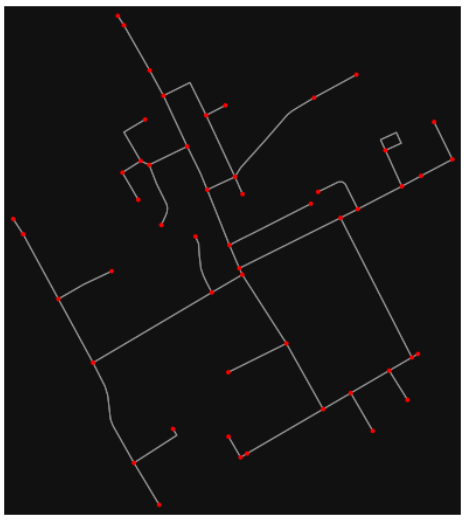
```
# same point again, but create network only of nodes within 500m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```



Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [42]:

```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [ ]:

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [43]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [44]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [45]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [46]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [47]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

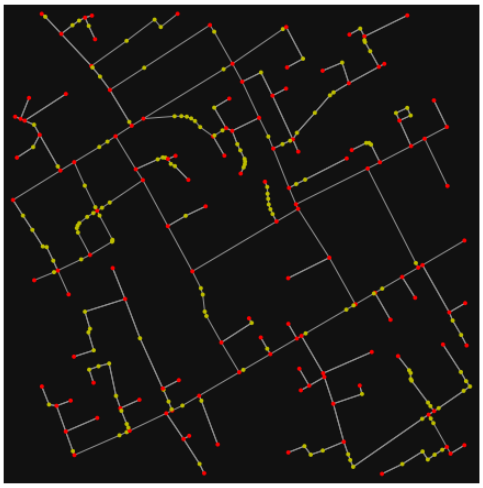
Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [48]:

```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (41.33323, 19.82686)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [49]:

```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```

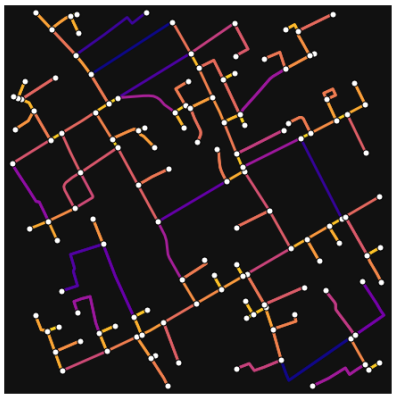


The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [50]:

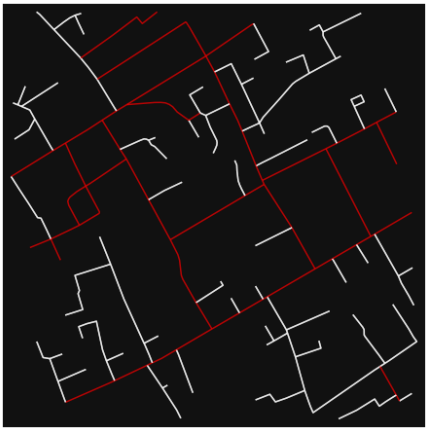
```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```

In [51]:
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [52]:
```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [53]:
```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [54]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/tirana_1st_zone_network.gpkg")
```

In [55]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/tirana_1st_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [56]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[56]:

2.3059701492537314

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.
In [57]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[57]:

90

In [58]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[58]:

223449.60236870352

In [59]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[59]:


```
{'n': 134,
 'm': 228,
 'k_avg': 3.4029850746268657,
 'edge_length_total': 9357.851999999997,
 'edge_length_avg': 41.043210526315775,
 'streets_per_node_avg': 2.3059701492537314,
 'streets_per_node_counts': {0: 0, 1: 44, 2: 9, 3: 77, 4: 4},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.3283582089552239,
  2: 0.06716417910447761,
  3: 0.5746268656716418,
  4: 0.029850746268656716},
 'intersection_count': 90,
 'street_length_total': 6022.533999999998,
 'street_segment_count': 143,
 'street_length_avg': 42.115622377622365,
 'circuity_avg': 1.0606111539639405,
 'self_loop_proportion': 0.006993006993006993,
 'clean_intersection_count': 55,
 'node_density_km': 599.6877979621239,
 'intersection_density_km': 402.77538669097873,
 'edge_density_km': 41879.027309966084,
 'street_density_km': 26952.538452328514,
 'clean_intersection_density_km': 246.14051408893147}
```


In [60]:

```
import pandas as pd
```

In [61]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[61]:

	value
n	134
m	228
k_avg	3.402985
edge_length_total	9357.852
edge_length_avg	41.043211
streets_per_node_avg	2.30597
streets_per_node_counts	{0: 0, 1: 44, 2: 9, 3: 77, 4: 4}

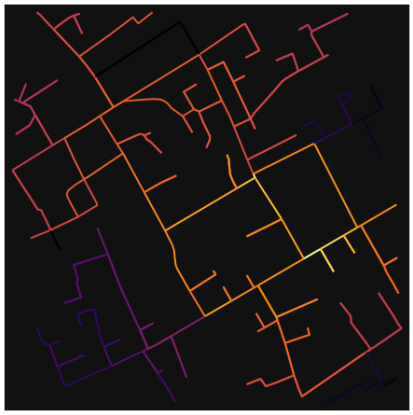
	value
streets_per_node_proportions	{0: 0.0, 1: 0.3283582089552239, 2: 0.067164179...
intersection_count	90
street_length_total	6022.534
street_segment_count	143
street_length_avg	42.115622
circuity_avg	1.061024
self_loop_proportion	0.006993
node_density_km	599.687798
intersection_density_km	402.775387
edge_density_km	41879.02731
street_density_km	26952.538452
0way_int_count	0
1way_int_count	44
2way_int_count	9
3way_int_count	77
4way_int_count	4
0way_int_prop	0.0
1way_int_prop	0.328358
2way_int_prop	0.067164
3way_int_prop	0.574627
4way_int_prop	0.029851

In [62]:
```
import networkx as nx
```

In [63]:
```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [64]:
```
# color edges in original graph with closeness centralities from line
graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```



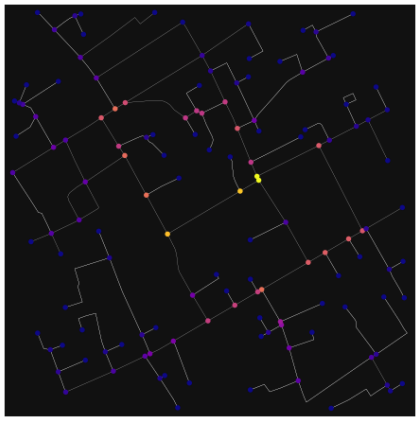
In [65]:
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
```

```
 max_node, max_bc = max(bc.items(), key=lambda x: x[1])
 max_node, max_bc
```

Out[65]:
(6952833317, 0.46958304853041694)

In [66]:

```
                                # add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 6: KAMPIONI URBAN TIRANA 2

In [1]:

```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```

In [5]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W414659023"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	POLYGON ((19.80743 41.32421, 19.80745 41.32415 ...	41.32427	41.32152	19.80756	19.80727	196106283	way	414659023	41.32192	19.80786	97, Rruga Myslym Shyri, Njësia Bashkiake Nr. 1...	building	house	0

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box*¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Tirana 2nd Zone
north, south, east, west = 41.3273, 41.32203, 19.80573, 19.81073

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters*¶

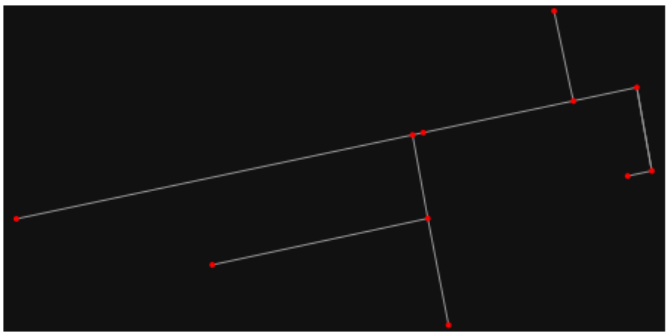
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (41.32411, 19.80756)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters*¶

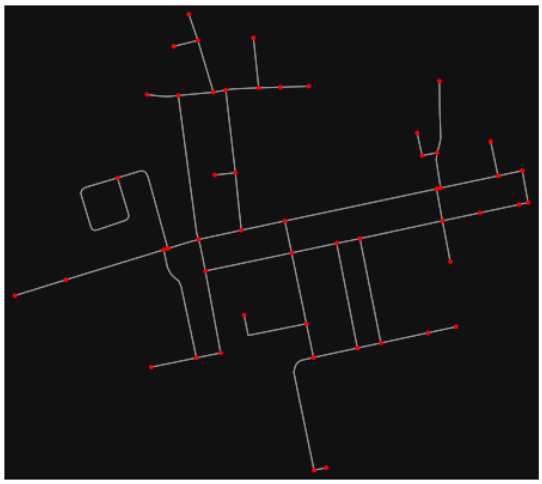
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```



Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:
```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [ ]:

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [10]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [11]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [12]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [13]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [14]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

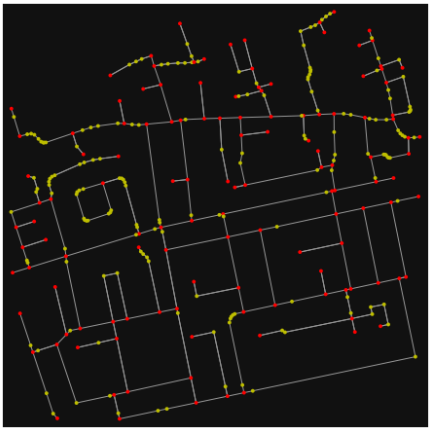
- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [15]:
```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (41.32411, 19.80756)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [16]:

```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [17]:

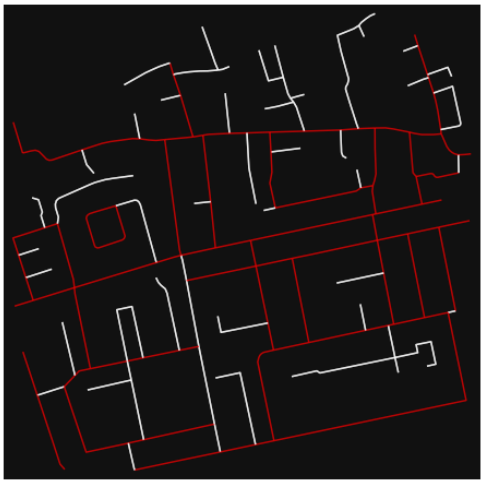
```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```

In [18]:

```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [19]:
```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [20]:
```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [21]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/tirana_1st_zone_network.gpkg")
```

In [22]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/tirana_1st_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [23]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[23]:

2.440944881889764

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[24]:

92

In [25]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[25]:

200921.08920273493

In [26]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[26]:

```
{'n': 127,
 'm': 202,
 'k_avg': 3.1811023622047245,
 'edge_length_total': 9536.056999999995,
 'edge_length_avg': 47.20820297029701,
 'streets_per_node_avg': 2.440944881889764,
 'streets_per_node_counts': {0: 0, 1: 35, 2: 7, 3: 79, 4: 6},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.2755905511811024,
  2: 0.05511811023622047,
  3: 0.6220472440944882,
  4: 0.047244094488188976},
 'intersection_count': 92,
 'street_length_total': 6986.805000000002,
 'street_segment_count': 147,
 'street_length_avg': 47.52928571428573,
 'circuity_avg': 1.0972262808156323,
 'self_loop_proportion': 0.006802721088435374,
 'clean_intersection_count': 49,
 'node_density_km': 632.0889484719719,
 'intersection_density_km': 457.8912067670978,
 'edge_density_km': 47461.702690541606,
 'street_density_km': 34773.875792352104,
 'clean_intersection_density_km': 243.8768383868238}
```

In [27]:

import pandas as pd

In [28]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[28]:

	value
n	127
m	202
k_avg	3.181102
edge_length_total	9536.057
edge_length_avg	47.208203
streets_per_node_avg	2.440945
streets_per_node_counts	{0: 0, 1: 35, 2: 7, 3: 79, 4: 6}
streets_per_node_proportions	{0: 0.0, 1: 0.2755905511811024, 2: 0.055118110...
intersection_count	92
street_length_total	6986.805
street_segment_count	147
street_length_avg	47.529286
circuity_avg	1.097663
self_loop_proportion	0.006803
node_density_km	632.088948
intersection_density_km	457.891207
edge_density_km	47461.702691
street_density_km	34773.875792
0way_int_count	0
1way_int_count	35
2way_int_count	7
3way_int_count	79
4way_int_count	6
0way_int_prop	0.0
1way_int_prop	0.275591
2way_int_prop	0.055118
3way_int_prop	0.622047
4way_int_prop	0.047244

In [29]:

```
import networkx as nx
```

In [30]:
```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [31]:
```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [32]:
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[32]:
(1854612498, 0.3895238095238095)

In [33]:
```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```

ANEKSI 7: KAMPIONI URBAN TIRANA 3

In [34]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

Out[34]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [35]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

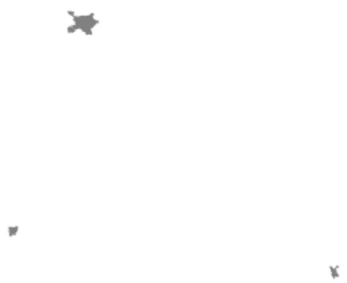
OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [36]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [37]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
```

```
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  pd.Int64Index,
```



In [38]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W51680778"], by_osmid=True)
```

Out[38]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.81514 41.31629, 19.81511 41.316...	41.316291	41.316128	19.81514	19.812572	114419405	way	51680778	41.316248	19.813694	Rruga Margarita Tutulani, Tirana e Re, Njësia ...	highway	residential	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)

- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [39]:
```
# define a bounding box in Tirana 5th Zone
north, south, east, west = 41.31851, 41.31351, 19.81080, 19.81580

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [40]:
```
# define a point at the corner of the interested zone
location_point = (41.31623, 19.81333)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

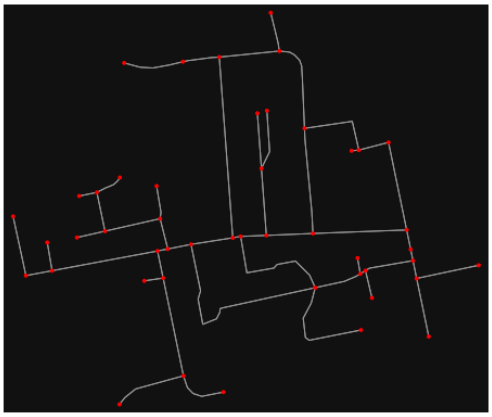
In [41]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

Note the plot above shows the network within 250m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way

and you cannot travel the wrong direction down them. Thus, the 250m takes into account only those nodes you can reach within 250m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 250m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [42]:

```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [ ]:

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [43]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [44]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [45]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [46]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [47]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [48]:
```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (41.31623, 19.81333)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [49]:

```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```

The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [50]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```

In [51]:

```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [52]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [53]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```

- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [54]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/tirana_7th_zone_network.gpkg")
```

In [55]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/tirana_7th_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [56]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[56]:

2.4065040650406506

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [57]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[57]:

91

In [58]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[58]:

199756.8623511094

In [59]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[59]:

```
{'n': 123,
 'm': 225,
 'k_avg': 3.658536585365854,
 'edge_length_total': 11036.64099999999,
 'edge_length_avg': 49.05173777777774,
 'streets_per_node_avg': 2.4065040650406506,
 'streets_per_node_counts': {0: 0, 1: 32, 2: 14, 3: 72, 4: 5},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.2601626016260163,
  2: 0.11382113821138211,
  3: 0.5853658536585366,
  4: 0.04065040650406504},
 'intersection_count': 91,
 'street_length_total': 6801.886999999999,
 'street_segment_count': 139,
 'street_length_avg': 48.934438848920855,
 'circuity_avg': 1.0951048653762074,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 50,
 'node_density_km': 615.7485582838445,
 'intersection_density_km': 455.55381141325086,
 'edge_density_km': 55250.37222801921,
 'street_density_km': 34050.830193980684,
 'clean_intersection_density_km': 250.30429198530265}
```

In [60]:

```
import pandas as pd
```

In [61]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[61]:

	value
n	123
m	225
k_avg	3.658537
edge_length_total	11036.641

	value
edge_length_avg	49.051738
streets_per_node_avg	2.406504
streets_per_node_counts	{0: 0, 1: 32, 2: 14, 3: 72, 4: 5}
streets_per_node_proportions	{0: 0.0, 1: 0.2601626016260163, 2: 0.113821138...
intersection_count	91
street_length_total	6801.887
street_segment_count	139
street_length_avg	48.934439
circuity_avg	1.095734
self_loop_proportion	0.0
node_density_km	615.748558
intersection_density_km	455.553811
edge_density_km	55250.372228
street_density_km	34050.830194
0way_int_count	0
1way_int_count	32
2way_int_count	14
3way_int_count	72
4way_int_count	5
0way_int_prop	0.0
1way_int_prop	0.260163
2way_int_prop	0.113821
3way_int_prop	0.585366
4way_int_prop	0.04065

In [62]:

```
import networkx as nx
```

In [63]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [64]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [65]:

```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[65]:

(258158441, 0.2951497087115567)

In [66]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```

ANEKSI 8: KAMPIONI URBAN TIRANA 4

In [34]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

Out[34]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [35]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

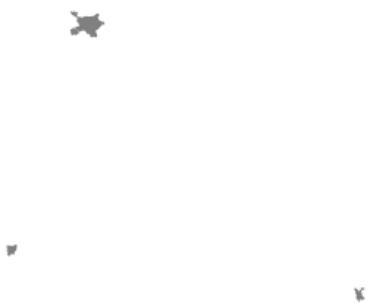
OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [36]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [37]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
   ]
albania_place = ox.geocode_to_gdf(place_names)
```

```
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```



In [38]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W173567512"], by_osmid=True)
```

Out[38]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.81756 41.34450, 19.81762 41.34 4...	41.345231	41.344497	19.820442	19.817565	142822648	way	173567512	41.344497	19.81762	Rruga Haxhi Xhediku, Njësia Bashkiake Nr. 8, T...	highway	residential	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [39]:
```
# define a bounding box in Tirana 2nd Zone
north, south, east, west = 41.372, 41.3422, 19.8165, 19.8215

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

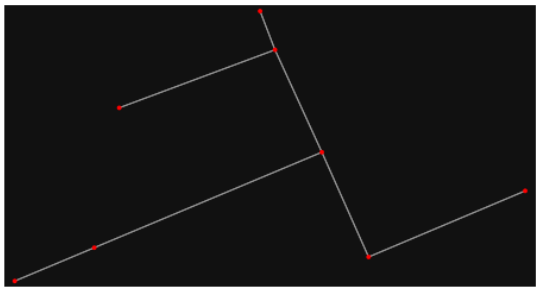
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [40]:
```
# define a point at the corner of the interested zone
location_point = (41.34481, 19.81889)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [41]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

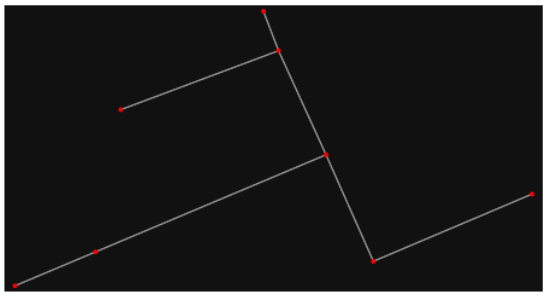


Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is

allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [42]:

```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [ ]:

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [43]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [44]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [45]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [46]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [47]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

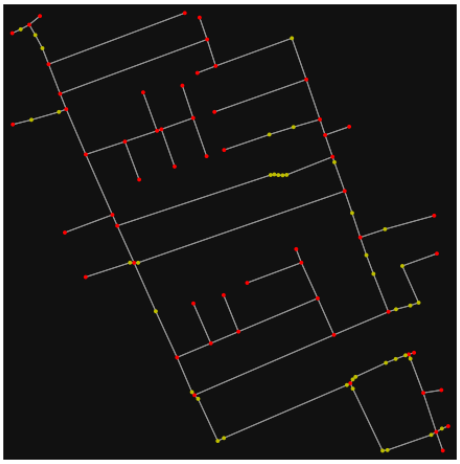
- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [48]:
```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (41.34481, 19.81889)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [49]:

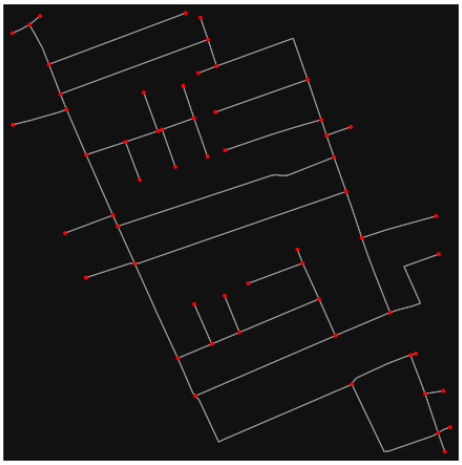
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

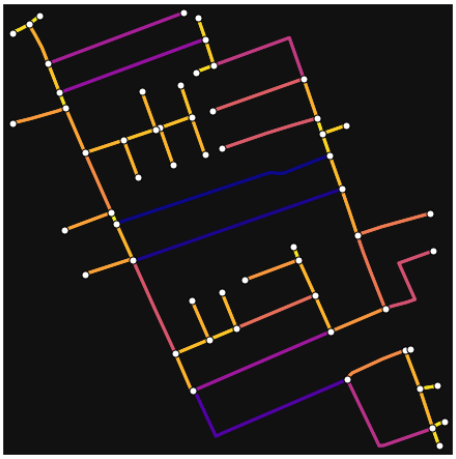
In [50]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```

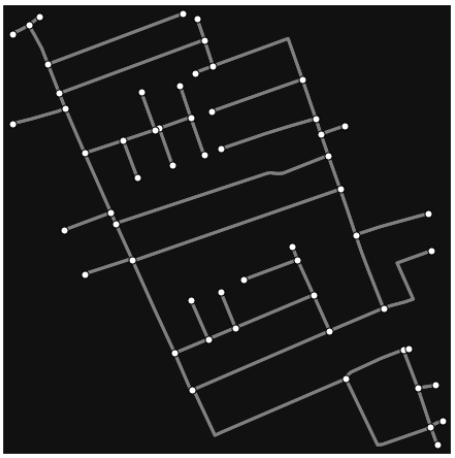


In [51]:

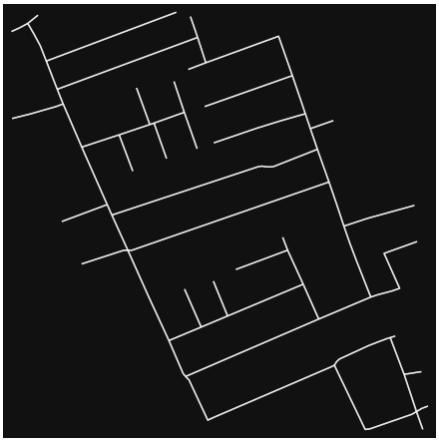
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [52]:
```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [53]:
```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [54]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/tirana_1st_zone_network.gpkg")
```

In [55]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/tirana_1st_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [56]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[56]:

2.310344827586207

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [57]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[57]:

40

In [58]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[58]:
165112.55689852944

In [59]:
```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[59]:
```
                                {'n': 58,
 'm': 124,
 'k_avg': 4.275862068965517,
 'edge_length_total': 8295.032,
 'edge_length_avg': 66.89541935483871,
 'streets_per_node_avg': 2.310344827586207,
 'streets_per_node_counts': {0: 0, 1: 18, 2: 7, 3: 30, 4: 3},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.3103448275862069,
  2: 0.1206896551724138,
  3: 0.5172413793103449,
  4: 0.05172413793103448},
```

```
 'intersection_count': 40,
 'street_length_total': 4147.515999999999,
 'street_segment_count': 62,
 'street_length_avg': 66.8954193548387,
 'circuity_avg': 1.039537950556496,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 29,
 'node_density_km': 351.2755243421257,
 'intersection_density_km': 242.2589823049143,
 'edge_density_km': 50238.65026266744,
 'street_density_km': 25119.325131333717,
 'clean_intersection_density_km': 175.63776217106286}
```

In [60]:

```
import pandas as pd
```

In [61]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[61]:

	value
n	58
m	124
k_avg	4.275862
edge_length_total	8295.032
edge_length_avg	66.895419
streets_per_node_avg	2.310345
streets_per_node_counts	{0: 0, 1: 18, 2: 7, 3: 30, 4: 3}
streets_per_node_proportions	{0: 0.0, 1: 0.3103448275862069, 2: 0.120689655...
intersection_count	40
street_length_total	4147.516
street_segment_count	62
street_length_avg	66.895419
circuity_avg	1.040454
self_loop_proportion	0.0
node_density_km	351.275524
intersection_density_km	242.258982
edge_density_km	50238.650263
street_density_km	25119.325131
0way_int_count	0
1way_int_count	18

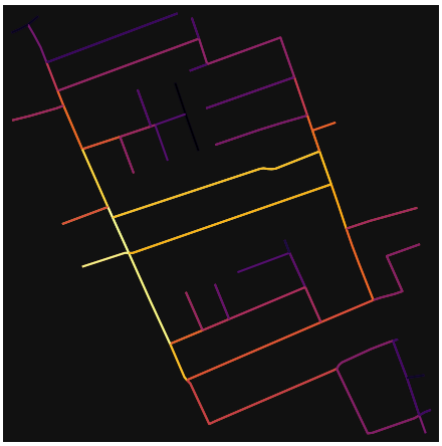
	value
2way_int_count	7
3way_int_count	30
4way_int_count	3
0way_int_prop	0.0
1way_int_prop	0.310345
2way_int_prop	0.12069
3way_int_prop	0.517241
4way_int_prop	0.051724

In [62]:
```
import networkx as nx
```

In [63]:
```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [64]:
```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

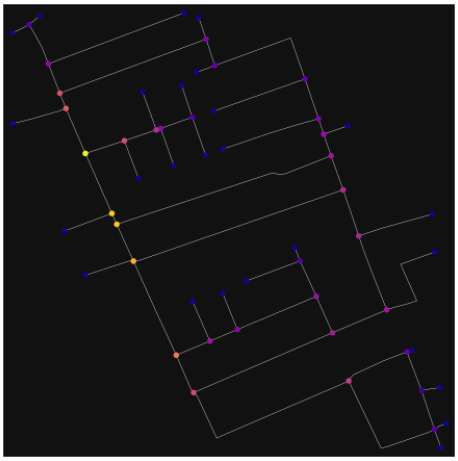


In [65]:
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[65]:
(1843585017, 0.47117794486215536)

In [66]:
```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
```

```
    edge_color="w",
)
```



ANEKSI 9: KAMPIONI URBAN TIRANA 5

In [1]:

```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

```
'1.2.0'
```

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

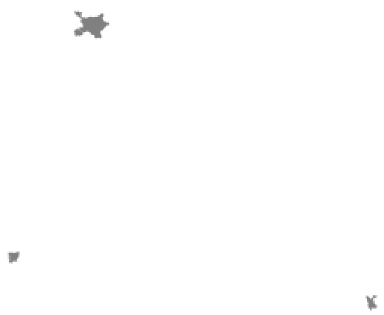
In [3]:

```
                        # get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
   ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```



In [5]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W472066493"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.84300 41.34360, 19.84342	41.344631	41.343601	19.84434	19.843004	205649249	way	472066493	41.343927	19.843421	Porcelani, Njësia Bashkiake Nr. 4, Shko	highway	living_street	0.1

geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
41.343...										zë, Tir...			

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box*¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Tirana 5th Zone
north, south, east, west = 41.3462, 41.3412, 19.8409, 19.8459

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (41.34393, 19.84344)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

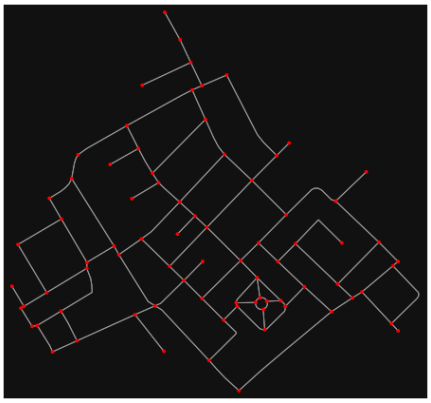
- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:
```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [ ]:

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [10]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [11]:
```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [12]:
```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [13]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [14]:
```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

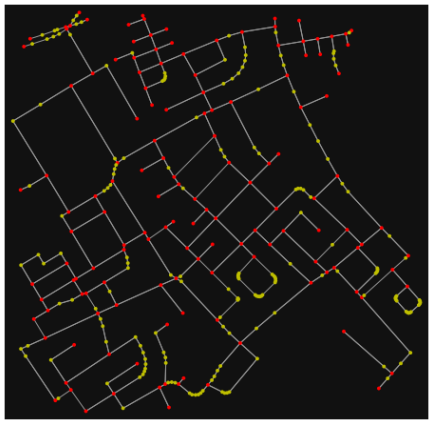
- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were

undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [15]:
```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (41.34393, 19.84344)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [16]:
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [17]:
```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```

In [18]:
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [19]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [20]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```

- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [21]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/tirana_5th_zone_network.gpkg")
```

In [22]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/tirana_5th_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [23]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[23]:
2.6595744680851063

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.
In [24]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[24]:

114

In [25]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[25]:
202851.51255576385

In [26]:
```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[26]:
```
                                {'n': 141,
 'm': 342,
 'k_avg': 4.851063829787234,
 'edge_length_total': 13771.099000000011,
 'edge_length_avg': 40.26637134502927,
 'streets_per_node_avg': 2.6595744680851063,
 'streets_per_node_counts': {0: 0, 1: 27, 2: 10, 3: 88, 4: 16},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.19148936170212766,
  2: 0.07092198581560284,
  3: 0.624113475177305,
  4: 0.11347517730496454},
 'intersection_count': 114,
 'street_length_total': 7267.103999999999,
```

'street_segment_count': 180,
'street_length_avg': 40.3728,
'circuity_avg': 1.112994174219338,
'self_loop_proportion': 0.011111111111111112,
'clean_intersection_count': 41,
'node_density_km': 695.0897147549695,
'intersection_density_km': 561.9874289508264,
'edge_density_km': 67887.58351611668,
'street_density_km': 35824.74642875671,
'clean_intersection_density_km': 202.1182858507358}

In [27]:
```
import pandas as pd
```

In [28]:
```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[28]:

	value
n	141
m	342
k_avg	4.851064
edge_length_total	13771.099
edge_length_avg	40.266371
streets_per_node_avg	2.659574
streets_per_node_counts	{0: 0, 1: 27, 2: 10, 3: 88, 4: 16}
streets_per_node_proportions	{0: 0.0, 1: 0.19148936170212766, 2: 0.07092198...
intersection_count	114
street_length_total	7267.104
street_segment_count	180
street_length_avg	40.3728
circuity_avg	1.113353
self_loop_proportion	0.011111
node_density_km	695.089715
intersection_density_km	561.987429
edge_density_km	67887.583516
street_density_km	35824.746429
0way_int_count	0
1way_int_count	27
2way_int_count	10
3way_int_count	88

	value
4way_int_count	16
0way_int_prop	0.0
1way_int_prop	0.191489
2way_int_prop	0.070922
3way_int_prop	0.624113
4way_int_prop	0.113475

In [29]:

```
import networkx as nx
```

In [30]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [31]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [32]:

```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[32]:

```
(1854591702, 0.30041109969167523)
```

In [33]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```

ANEKSI 10: KAMPIONI URBAN TIRANA 6

In [34]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

Out[34]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [35]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

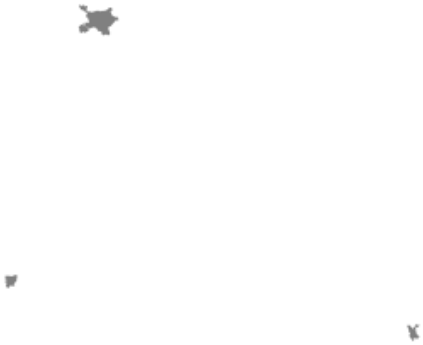
OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [36]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [37]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```



In [38]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W175334841"], by_osmid=True)
```

Out[38]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.8	41.3349 52	41.3342 48	19.828657	19.827542	142483649	way	175334841	41.334675	19.828023	Rruga Haxhi	highway	living_street	0.1

geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
28664133495, 19.82827 41.334...										Sina, Medreseja, Njësia Bashkiake ...			

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box*¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [39]:
```
# define a bounding box in Tirana 5th Zone
north, south, east, west = 41.33735, 41.33235, 19.82592, 19.83092

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [40]:
```
# define a point at the corner of the interested zone
location_point = (41.33481, 19.82822)
```

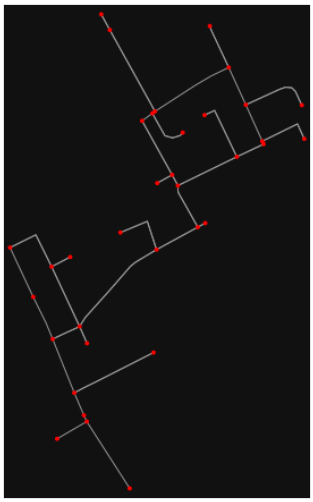
```
# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [41]:

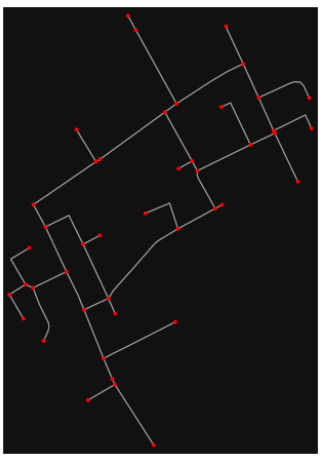
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```



Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [42]:

```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [ ]:

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [43]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [44]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [45]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [46]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

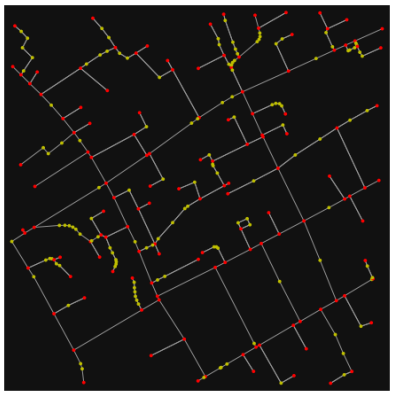
In [47]:
```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

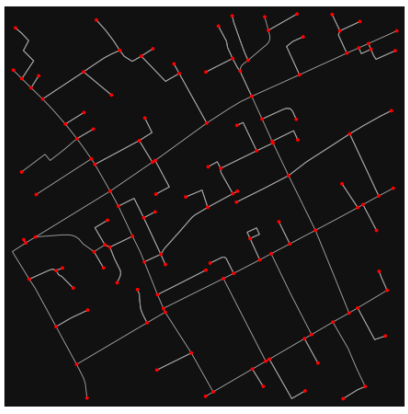
In [48]:
```
                                # create a network around some (lat, lng) point but do not simplify it yet
location_point = (41.33481, 19.82822)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [49]:
```
                                # turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```

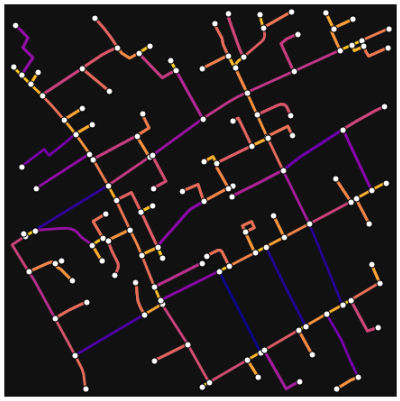


The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

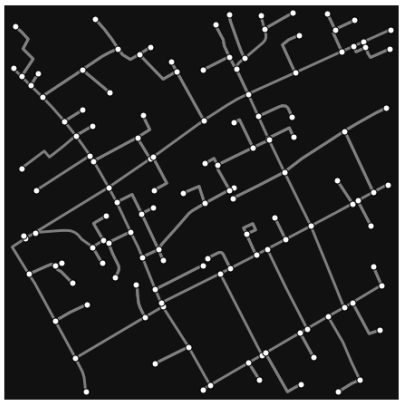
In [50]:
```
                                # simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



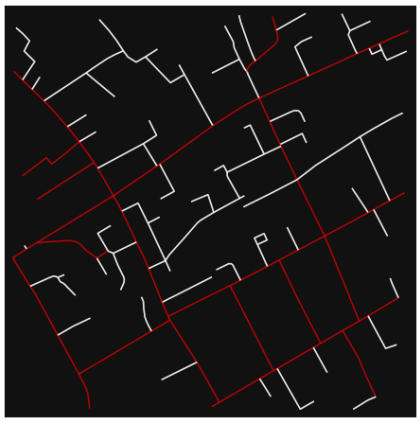
In [51]:
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [52]:
```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [53]:
```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [54]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/tirana_6th_zone_network.gpkg")
```

In [55]:
```
# save street network as GraphML file to work with later in OSMnx or
networkx or gephi
ox.save_graphml(G, filepath="./data/tirana_6th_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [56]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[56]:
2.3484848484848486

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [57]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[57]:
93

In [58]:
```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[58]:
226414.96671061593

In [59]:
```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[59]:

```
{'n': 132,
 'm': 219,
 'k_avg': 3.3181818181818183,
 'edge_length_total': 9248.676000000005,
 'edge_length_avg': 42.231397260273994,
 'streets_per_node_avg': 2.3484848484848486,
 'streets_per_node_counts': {0: 0, 1: 39, 2: 14, 3: 73, 4: 6},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.29545454545454547,
  2: 0.10606060606060606,
  3: 0.553030303030303,
  4: 0.045454545454545456},
 'intersection_count': 93,
 'street_length_total': 6135.543999999998,
 'street_segment_count': 144,
 'street_length_avg': 42.60794444444443,
```

```
 'circuity_avg': 1.051143199461809,
 'self_loop_proportion': 0.006944444444444444,
 'clean_intersection_count': 53,
 'node_density_km': 583.0003286342418,
 'intersection_density_km': 410.7502315377613,
 'edge_density_km': 40848.34202599718,
 'street_density_km': 27098.667942044314,
 'clean_intersection_density_km': 234.0834652849607}
```

In [60]:

```
import pandas as pd
```

In [61]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]
# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[61]:

	value
n	132
m	219
k_avg	3.318182
edge_length_total	9248.676
edge_length_avg	42.231397
streets_per_node_avg	2.348485
streets_per_node_counts	{0: 0, 1: 39, 2: 14, 3: 73, 4: 6}
streets_per_node_proportions	{0: 0.0, 1: 0.29545454545454547, 2: 0.10606060...
intersection_count	93
street_length_total	6135.544
street_segment_count	144
street_length_avg	42.607944
circuity_avg	1.051589
self_loop_proportion	0.006944
node_density_km	583.000329
intersection_density_km	410.750232
edge_density_km	40848.342026
street_density_km	27098.667942
0way_int_count	0
1way_int_count	39
2way_int_count	14
3way_int_count	73
4way_int_count	6
0way_int_prop	0.0

	value
1way_int_prop	0.295455
2way_int_prop	0.106061
3way_int_prop	0.55303
4way_int_prop	0.045455

In [62]:

```
import networkx as nx
```

In [63]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [64]:

```
# color edges in original graph with closeness centralities
from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [65]:

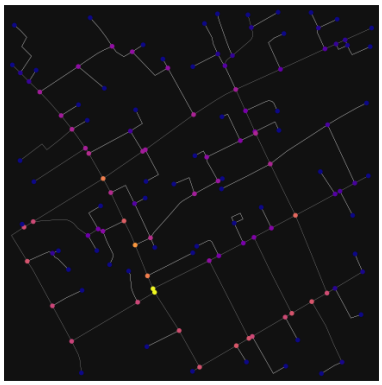
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[65]:

```
(1851757540, 0.3470346447445684)
```

In [66]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 11: KAMPIONI URBAN TIRANA 7

In [1]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

```
'1.2.0'
```

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

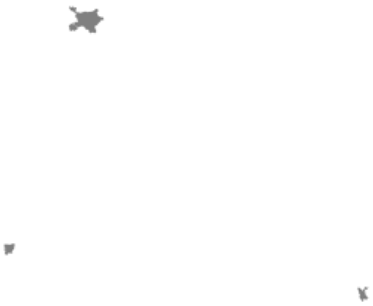
- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Tirana, project it, and plot it
city = ox.geocode_to_gdf("Tiranë, Tirana County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  pd.Int64Index,
```



In [5]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W174799623"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.82971 41.34678, 19.83002	41.347306	41.346782	19.833006	19.829711	143216406	way	174799623	41.347054	19.831428	Rruga Ram Sadria, Allias, Njësia Bash	highway	tertiary	0.1

geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
41.346...										kiake Nr....			

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box*¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Tirana 5th Zone
north, south, east, west = 41.3492, 41.3442, 19.8280, 19.8330

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

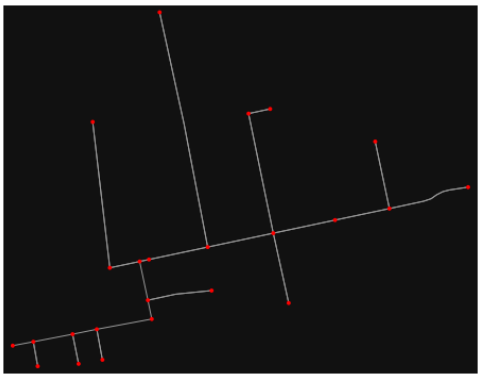
In [7]:
```
# define a point at the corner of the interested zone
location_point = (41.34694, 19.83074)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:

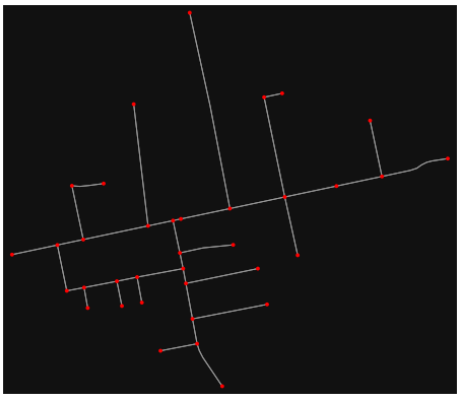
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```



Note the plot above shows the network within 250m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 250m takes into account only those nodes you can reach within 250m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 250m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [ ]:

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [10]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Tiranë, Tirana County, Albania", network_type="drive")
```

In [11]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Tiranë, Tirana County, Albania",
    {"city": "Tiranë", "state": "Albania"},
    "Tiranë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [12]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [13]:

```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [14]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

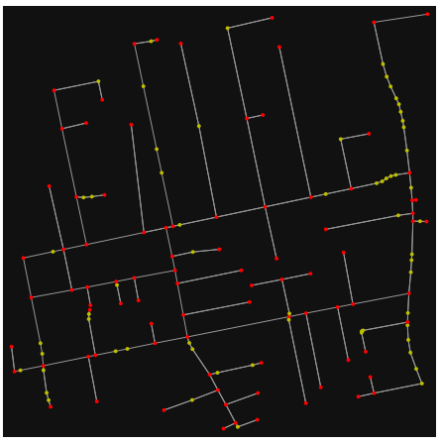
Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were

undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [15]:
```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (41.34694, 19.83074)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

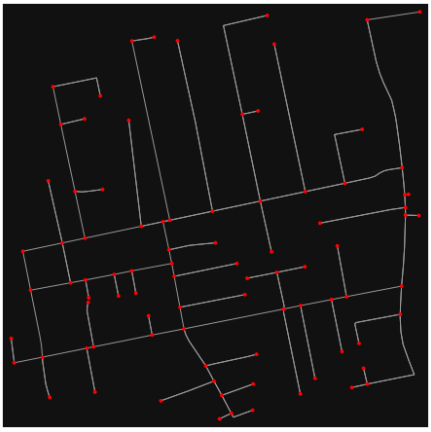
In [16]:

```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```

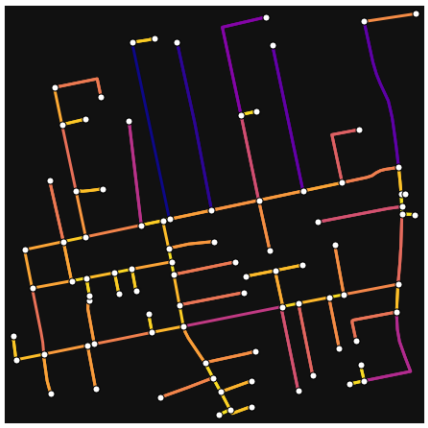


The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [17]:
```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```

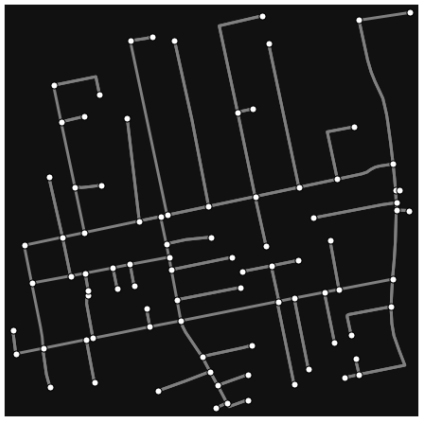


In [18]:
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



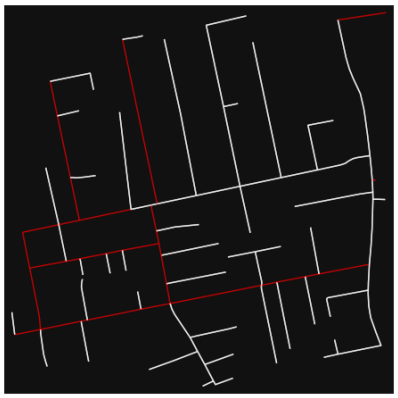
In [19]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [20]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [21]:

```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/tirana_8th_zone_network.gpkg")
```

In [22]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/tirana_8th_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [23]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[23]:

2.227272727272727

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[24]:

53

In [25]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[25]:
203376.95764670984

In [26]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[26]:

```
{'n': 88,
 'm': 151,
 'k_avg': 3.4318181818181817,
 'edge_length_total': 8690.673000000003,
 'edge_length_avg': 57.55412582781459,
 'streets_per_node_avg': 2.227272727272727,
 'streets_per_node_counts': {0: 0, 1: 35, 2: 5, 3: 41, 4: 7},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.3977272727272727,
  2: 0.056818181818181816,
  3: 0.4659090909090909,
  4: 0.07954545454545454},
 'intersection_count': 53,
 'street_length_total': 5110.607999999999,
 'street_segment_count': 91,
 'street_length_avg': 56.16052747252746,
 'circuity_avg': 1.0302360263387815,
```

```
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 34,
 'node_density_km': 432.6940525527309,
 'intersection_density_km': 260.599827105622,
 'edge_density_km': 42731.846815688645,
 'street_density_km': 25128.746437822803,
 'clean_intersection_density_km': 167.17724757719148}
```

In [27]:

```
import pandas as pd
```

In [28]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[28]:

	value
n	88
m	151
k_avg	3.431818
edge_length_total	8690.673
edge_length_avg	57.554126
streets_per_node_avg	2.227273
streets_per_node_counts	{0: 0, 1: 35, 2: 5, 3: 41, 4: 7}
streets_per_node_proportions	{0: 0.0, 1: 0.3977272727272727, 2: 0.056818181...
intersection_count	53
street_length_total	5110.608
street_segment_count	91
street_length_avg	56.160527
circuity_avg	1.030347
self_loop_proportion	0.0
node_density_km	432.694053
intersection_density_km	260.599827
edge_density_km	42731.846816
street_density_km	25128.746438
0way_int_count	0
1way_int_count	35
2way_int_count	5
3way_int_count	41
4way_int_count	7

	value
0way_int_prop	0.0
1way_int_prop	0.397727
2way_int_prop	0.056818
3way_int_prop	0.465909
4way_int_prop	0.079545

In [29]:

```
import networkx as nx
```

In [30]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [31]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [32]:

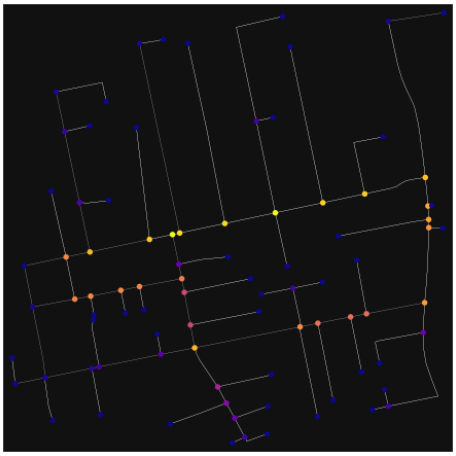
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[32]:

```
(1854582810, 0.39067094359796845)
```

In [33]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 12: KAMPIONI URBAN FIER 1

In [35]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

Out[35]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [36]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

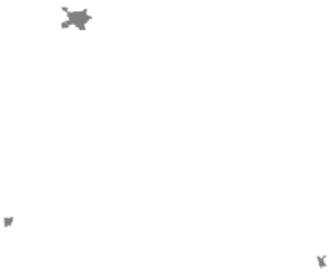
OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [37]:
```
# get the boundary polygon for Fier, project it, and plot it
city = ox.geocode_to_gdf("Fier, Fier, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [38]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```



In [39]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf([ "W176156310"], by_osmid=True)
```

Out[39]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.55960 40.73301,	40.733008	40.732515	19.563903	19.559605	142706245	way	176156310	40.732764	19.561684	Rruga Thoma Kopaçe, 29	highway	residential	0.1

geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
19.55980 40.732...										Nënt ori, Fier, Bash kia ...			

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [40]:
```
# define a bounding box in Fieri
north, south, east, west = 40.73549, 40.73049, 19.55935, 19.56435

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

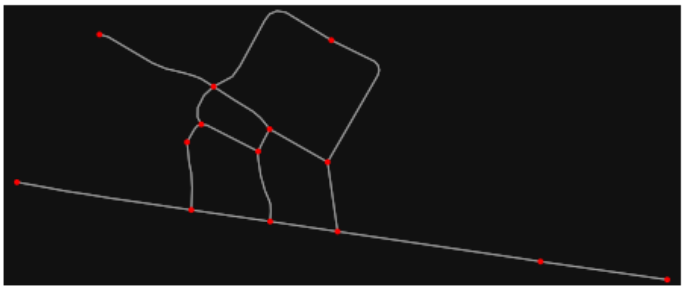
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [41]:
```
# define a point at the corner of the interested zone
location_point = (40.73270, 19.56222)

# create network from point, inside bounding box of N, S, E, W each 1000m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

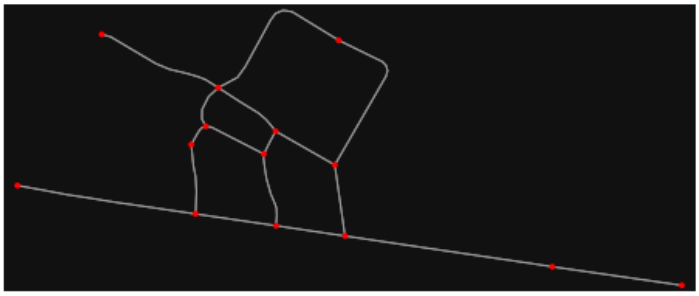
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [42]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```



Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [43]:
```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [44]:
```
# network from address, including only nodes within 1km along the network from the address
G = ox.graph_from_address(
    address="Rruga Thoma Kopaçe, 29 Nëntori, Sheq i Vogël, Fier, Bashkia Fier, Fier County, Southern
 Albania, 9304, Albania",
    dist=250,
    dist_type="network",
    network_type="drive",
)
```

```
# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [45]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Fier, Fier, Albania", network_type="drive")
```

In [46]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Fier, Fier, Albania",
    {"city": "Fier", "state": "Albania"},
    "Fier, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [47]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [48]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [49]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

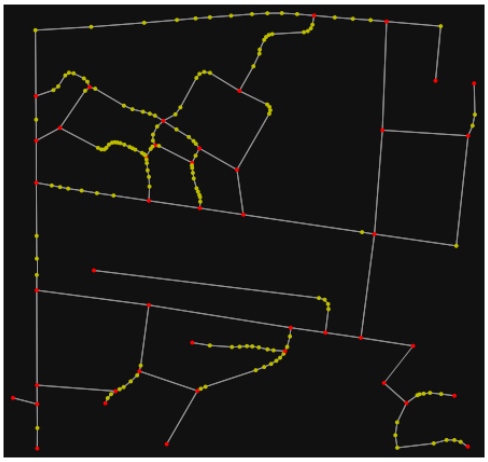
Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [50]:

```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.73270, 19.56222)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [51]:

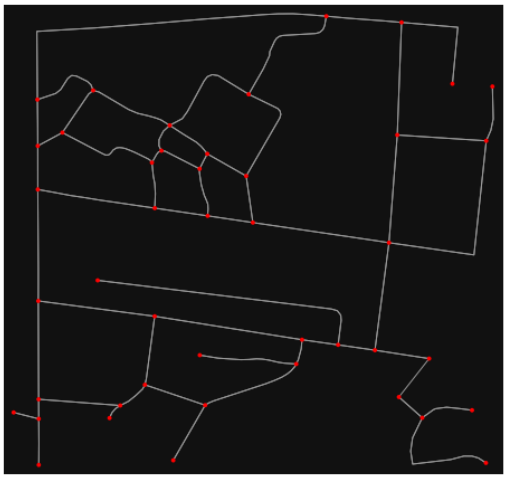
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [52]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



In [53]:

```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [54]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [55]:
```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```

- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [56]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/comunists_zone_network.gpkg")
```

In [57]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/comunists_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [58]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[58]:

2.75

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [59]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[59]:

40

In [60]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[60]:

203391.186407108

In [61]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[61]:

```
{'n': 44,
 'm': 111,
 'k_avg': 5.045454545454546,
 'edge_length_total': 9175.076999999997,
 'edge_length_avg': 82.65835135135133,
 'streets_per_node_avg': 2.75,
 'streets_per_node_counts': {0: 0, 1: 4, 2: 6, 3: 31, 4: 3},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.09090909090909091,
  2: 0.13636363636363635,
  3: 0.7045454545454546,
  4: 0.06818181818181818},
 'intersection_count': 40,
 'street_length_total': 4612.749999999999,
 'street_segment_count': 56,
 'street_length_avg': 82.3705357142857,
 'circuity_avg': 1.0786256193121035,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 32,
 'node_density_km': 216.33189115643168,
 'intersection_density_km': 196.6653555967561,
 'edge_density_km': 45110.49452081544,
 'street_density_km': 22679.202975723412,
 'clean_intersection_density_km': 157.33228447740487}
```

In [62]:

```
import pandas as pd
```

In [63]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
```

```
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[63]:

	value
n	44
m	111
k_avg	5.045455
edge_length_total	9175.077
edge_length_avg	82.658351
streets_per_node_avg	2.75
streets_per_node_counts	{0: 0, 1: 4, 2: 6, 3: 31, 4: 3}
streets_per_node_proportions	{0: 0.0, 1: 0.09090909090909091, 2: 0.13636363...
intersection_count	40
street_length_total	4612.75
street_segment_count	56
street_length_avg	82.370536
circuity_avg	1.079511
self_loop_proportion	0.0
node_density_km	216.331891
intersection_density_km	196.665356
edge_density_km	45110.494521
street_density_km	22679.202976
0way_int_count	0
1way_int_count	4
2way_int_count	6
3way_int_count	31
4way_int_count	3
0way_int_prop	0.0
1way_int_prop	0.090909
2way_int_prop	0.136364
3way_int_prop	0.704545
4way_int_prop	0.068182

In [64]:

```
import networkx as nx
```

In [65]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [66]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [67]:

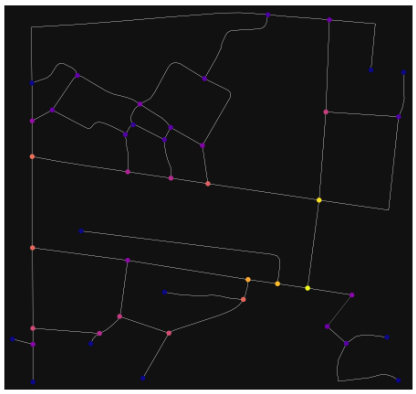
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[67]:

(1867120452, 0.31561461794019935)

In [68]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 13: KAMPIONI URBAN FIER 2

In [1]:

```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

```
'1.2.0'
```

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:

```
# get the boundary polygon for Fier, project it, and plot it
city = ox.geocode_to_gdf("Fier, Fier, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
```

```
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```

In [5]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf([ "W176158030"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.54938 40.73027, 19.54814 40.730...	40.73709	40.73269	19.54937	19.54548	143522903	way	176158030	40.73288	19.54602	Rruga Sokrat Plaku, Afrim i Ri, Fier, Bashkia ...	highway	residential	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads

- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Fieri
north, south, east, west = 40.7359, 40.7309, 19.5436, 19.5486

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

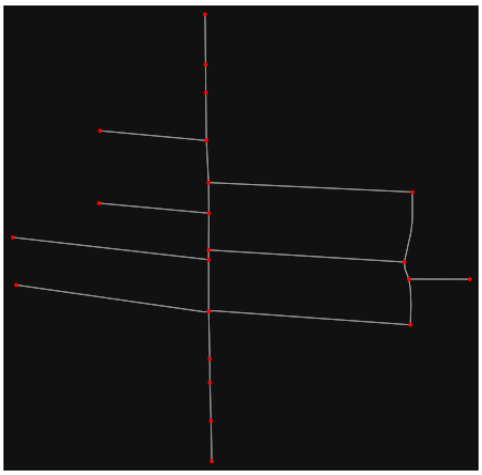
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (40.73373, 19.54603)

# create network from point, inside bounding box of N, S, E, W each 1000m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

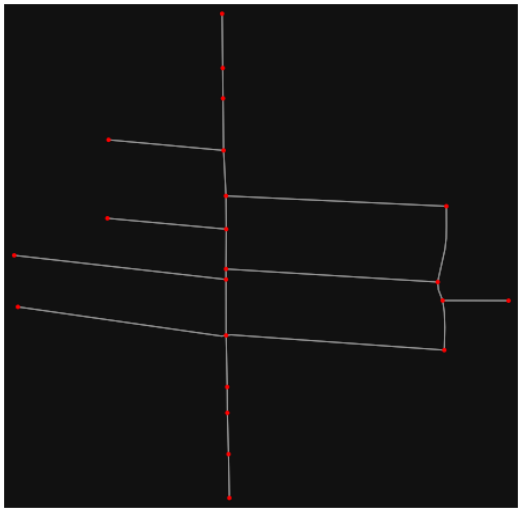


Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those

nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:

```
# network from address, including only nodes within 1km along the network from the address
G = ox.graph_from_address(
    address="Rruga Sokrat Plaku, Afrim i Ri, Qendër, Fier, Bashkia Fier, Fier County, Southern Albania
, 9301, Albania",
    dist=250,
    dist_type="network",
    network_type="drive",
)

# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Fier, Fier, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Fier, Fier, Albania",
    {"city": "Fier", "state": "Albania"},
```

```
    "Fier, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [13]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:

```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [15]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

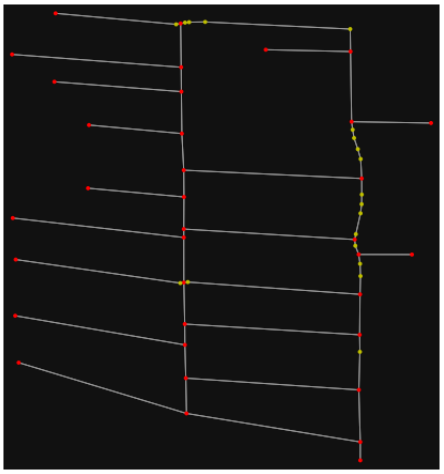
Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [16]:

```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.73373, 19.54603)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [17]:

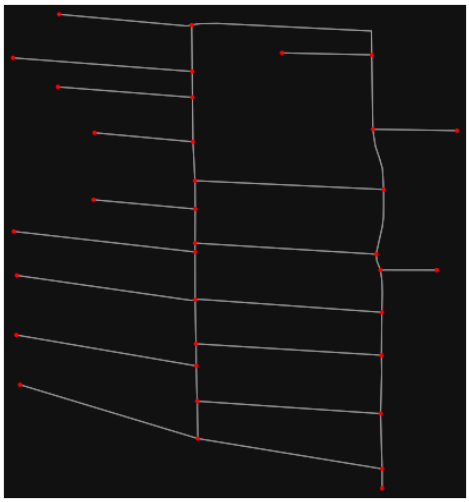
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

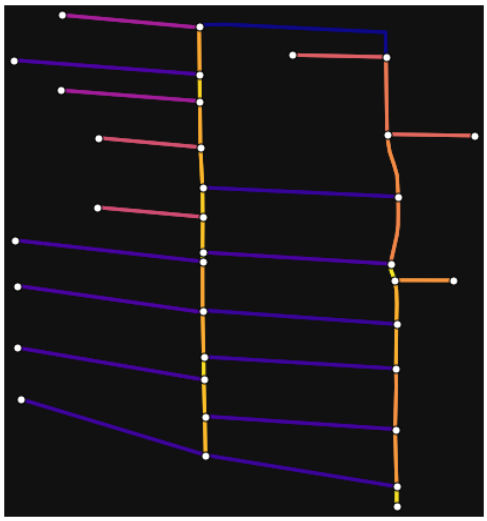
In [18]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



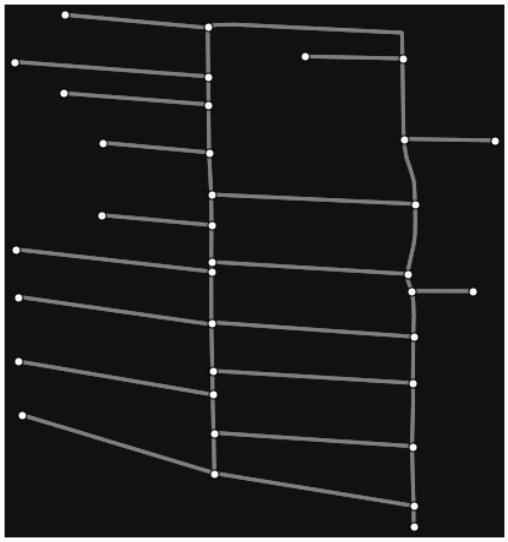
In [19]:

```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



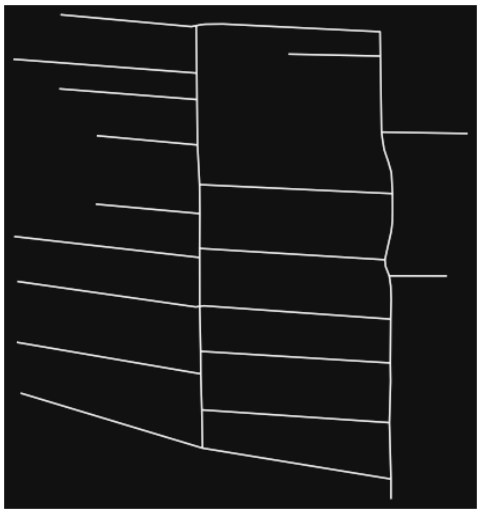
In [20]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [21]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/comunists_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/comunists_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

2.4857142857142858

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

24

In [26]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[26]:

152538.97197855785

In [27]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[27]:

```
{'n': 35,
 'm': 80,
 'k_avg': 4.571428571428571,
 'edge_length_total': 7237.392000000001,
 'edge_length_avg': 90.46740000000001,
 'streets_per_node_avg': 2.4857142857142858,
 'streets_per_node_counts': {0: 0, 1: 11, 2: 0, 3: 20, 4: 4},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.3142857142857143,
  2: 0.0,
  3: 0.5714285714285714,
  4: 0.11428571428571428},
 'intersection_count': 24,
 'street_length_total': 3618.696,
 'street_segment_count': 40,
 'street_length_avg': 90.4674,
 'circuity_avg': 1.0046320017297994,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 18,
 'node_density_km': 229.44955997815362,
 'intersection_density_km': 157.33684112787677,
 'edge_density_km': 47446.18313684026,
 'street_density_km': 23723.091568420128,
 'clean_intersection_density_km': 118.00263084590758}
```

In [28]:

```
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	35
m	80
k_avg	4.571429
edge_length_total	7237.392

	value
edge_length_avg	90.4674
streets_per_node_avg	2.485714
streets_per_node_counts	{0: 0, 1: 11, 2: 0, 3: 20, 4: 4}
streets_per_node_proportions	{0: 0.0, 1: 0.3142857142857143, 2: 0.0, 3: 0.5...
intersection_count	24
street_length_total	3618.696
street_segment_count	40
street_length_avg	90.4674
circuity_avg	1.006058
self_loop_proportion	0.0
node_density_km	229.44956
intersection_density_km	157.336841
edge_density_km	47446.183137
street_density_km	23723.091568
0way_int_count	0
1way_int_count	11
2way_int_count	0
3way_int_count	20
4way_int_count	4
0way_int_prop	0.0
1way_int_prop	0.314286
2way_int_prop	0.0
3way_int_prop	0.571429
4way_int_prop	0.114286

In [30]:

```
import networkx as nx
```

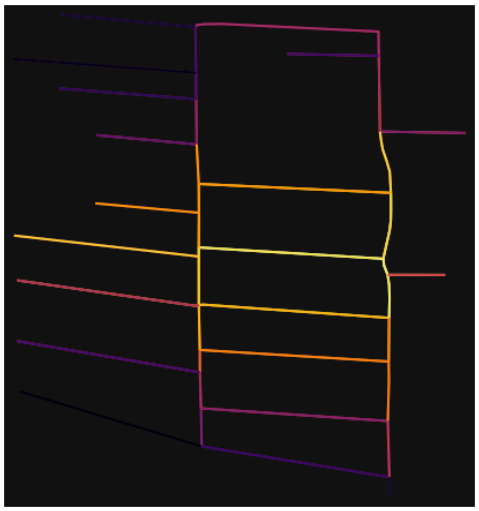
In [31]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```



In [33]:

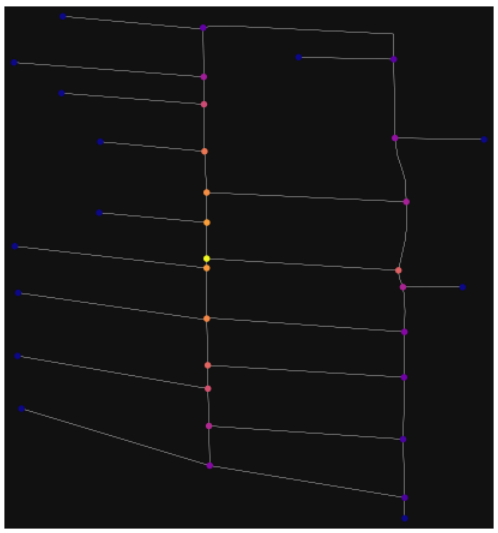
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

(1867132908, 0.4670231729055258)

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 14: KAMPIONI URBAN FIER 3

In [1]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

```
'1.2.0'
```

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Fier, project it, and plot it
city = ox.geocode_to_gdf("Fier, Fier, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
```

```
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```

In [5]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf([ "W176153448"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.55003 40.73646, 19.55127 40.736...	40.736551	40.73646	19.559575	19.550031	142715598	way	176153448	40.736508	19.555018	Rruga Nestor Pogaçe, Mbrostar, Fier, Bashkia F...	highway	tertiary	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Fieri
north, south, east, west = 40.73892, 40.73392, 19.55030, 19.55530

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

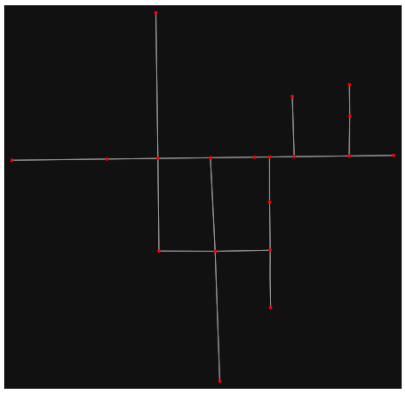
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (40.73646, 19.55284)

# create network from point, inside bounding box of N, S, E, W each 1000m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

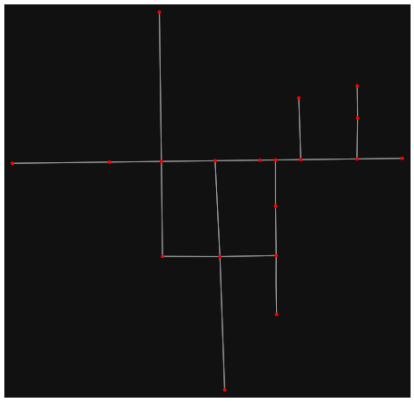


Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those

nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

```
# create network only of nodes within 250m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:

```
# network from address, including only nodes within 1km along the network from the address
G = ox.graph_from_address(
    address="Rruga Nestor Pogaçe, 8 Shkurti, Sheq i Vogël, Fier, Bashkia Fier, Fier County, Southern A
lbania, 9304, Albania",
    dist=250,
    dist_type="network",
    network_type="drive",
)

# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Fier, Fier, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Fier, Fier, Albania",
    {"city": "Fier", "state": "Albania"},
    "Fier, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [13]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:

```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [15]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

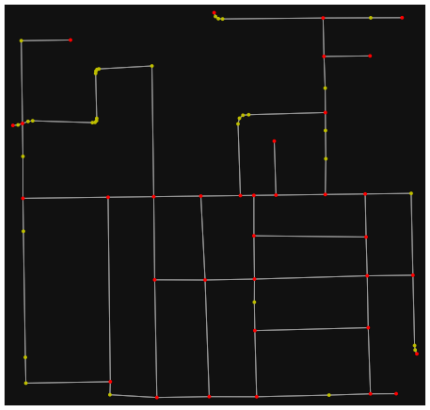
Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [16]:

```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.73646, 19.55284)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [17]:

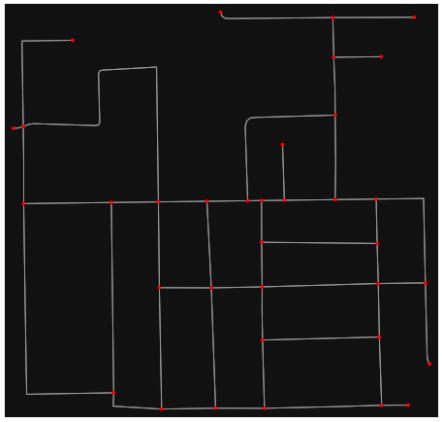
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [18]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



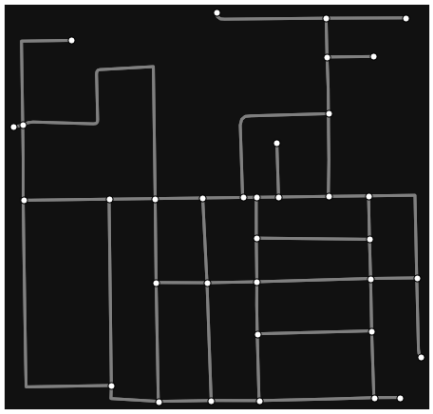
In [19]:

```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [20]:

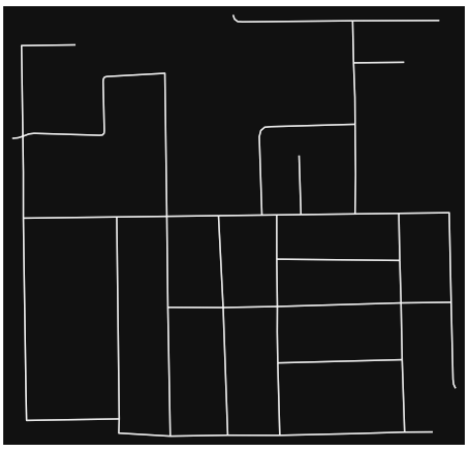
```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
```

```
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [21]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:

```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/comunists_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/comunists_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

3.0

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

32

In [26]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[26]:

201009.68128104522

In [27]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[27]:

```
{'n': 35,
 'm': 94,
 'k_avg': 5.371428571428571,
 'edge_length_total': 9202.996000000001,
 'edge_length_avg': 97.90421276595745,
 'streets_per_node_avg': 3.0,
 'streets_per_node_counts': {0: 0, 1: 3, 2: 3, 3: 20, 4: 9},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.08571428571428572,
  2: 0.08571428571428572,
  3: 0.5714285714285714,
  4: 0.2571428571428571},
 'intersection_count': 32,
 'street_length_total': 4601.4980000000005,
 'street_segment_count': 47,
 'street_length_avg': 97.90421276595745,
 'circuity_avg': 1.1052450855971403,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 29,
 'node_density_km': 174.1209665969478,
 'intersection_density_km': 159.19631231720942,
 'edge_density_km': 45783.84454593841,
 'street_density_km': 22891.922272969205,
 'clean_intersection_density_km': 144.27165803747104}
```

In [28]:

```
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	35
m	94
k_avg	5.371429
edge_length_total	9202.996
edge_length_avg	97.904213
streets_per_node_avg	3.0
streets_per_node_counts	{0: 0, 1: 3, 2: 3, 3: 20, 4: 9}
streets_per_node_proportions	{0: 0.0, 1: 0.08571428571428572, 2: 0.08571428...
intersection_count	32
street_length_total	4601.498
street_segment_count	47
street_length_avg	97.904213
circuity_avg	1.105587
self_loop_proportion	0.0
node_density_km	174.120967
intersection_density_km	159.196312
edge_density_km	45783.844546
street_density_km	22891.922273
0way_int_count	0
1way_int_count	3
2way_int_count	3
3way_int_count	20
4way_int_count	9
0way_int_prop	0.0
1way_int_prop	0.085714
2way_int_prop	0.085714
3way_int_prop	0.571429
4way_int_prop	0.257143

In [30]:

```
import networkx as nx
```

In [31]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [33]:

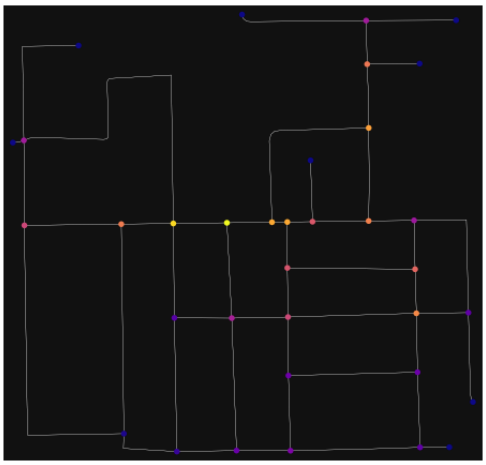
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

(1867132982, 0.3315508021390374)

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



In [ ]:

ANEKSI 15: KAMPIONI URBAN FIER 4

In [1]:

```
import geopandas as gpd

import osmnx as ox


%matplotlib inline

ox.__version__
```

C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas_compat.py:111: UserWarning: The Shapely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compiled with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.

  warnings.warn(

Out[1]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off

ox.settings.use_cache = False


# turn it back on and turn on/off logging to your console
```

```
ox.settings.use_cache = True

ox.settings.log_console = False
```

Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

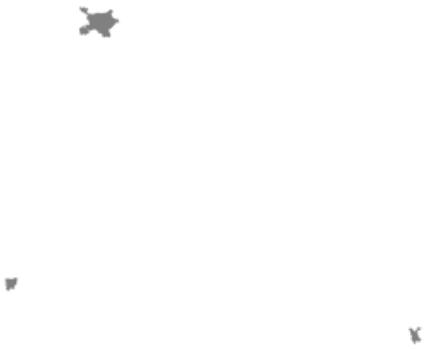
In [3]:

```
# get the boundary polygon for Fier, project it, and plot it

city = ox.geocode_to_gdf("Fier, Fier, Albania")

city_proj = ox.project_gdf(city)

ax = city_proj.plot(fc="gray", ec="none")

_ = ax.axis("off")
```

In [4]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot

place_names = [

    "Fier, Fier, Albania",

    "Korçë, Korçë County, Albania",

    "Tiranë, Tirana County, Albania",

  ]

albania_place = ox.geocode_to_gdf(place_names)

albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")

albania_place = ox.project_gdf(albania_place)

ax = albania_place.plot(fc="gray", ec="none")

_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning:
pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use
pandas.Index with the appropriate dtype instead.

  pd.Int64Index,
```



In [5]:

if you know the OSM ID of the place(s) you want, you can query it directly

ox.geocode_to_gdf(["W244286240"], by_osmid=True)

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (19.57034 40.73415, 19.57015 40.734 ...	40.734638	40.734151	19.570336	19.559593	161427433	way	244286240	40.734513	19.564746	Rruga Skënderbeu, Liri Gero, Sheq i Vogël, Fie...	highway	tertiary	0.1

Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

a bounding box

a lat-long point plus a distance

an address plus a distance

a place name or list of place names (to automatically geocode and get the boundary of)

a polygon of the desired street network's boundaries

a .osm formatted xml file

You can also specify several different network types:

'drive' - get drivable public streets (but not service roads)

'drive_service' - get drivable streets, including service roads

'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)

'bike' - get all streets and paths that cyclists can use

'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)

'all_private' - download all OSM streets and paths, including private-access ones

Method #1, pass a bounding box¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:

```
# define a bounding box in Fieri

north, south, east, west = 40.73737, 40.73237, 19.56618, 19.57118


# create network from that bounding box

G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

Method #2, pass a lat-lng point and bounding box distance in meters

This creates a bounding box n meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:

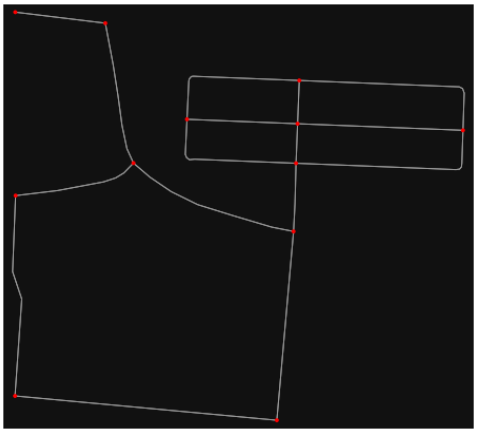
```
# define a point at the corner of the interested zone

location_point = (40.7355, 19.5684)


# create network from point, inside bounding box of N, S, E, W each 1000m from point

G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

Method #3, pass a lat-lng point and network distance in meters

This creates a bounding box n meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than n meters from the point along the network.

In [8]:

```
# same point again, but create network only of nodes within 250m along the network from
point

G = ox.graph_from_point(location_point, dist=250, dist_type="network")

fig, ax = ox.plot_graph(G, node_color="r")
```



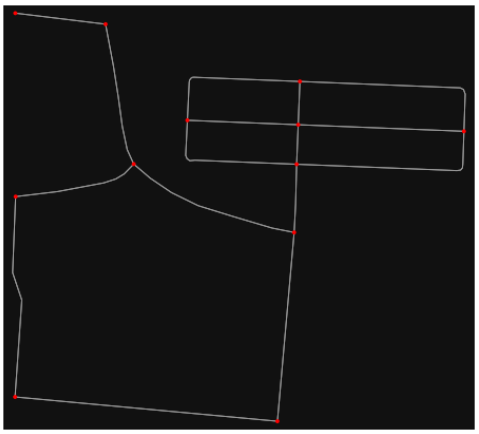
Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

create network only of nodes within 250m walking along the network from point

G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")

fig, ax = ox.plot_graph(G, node_color="r")



Method #4, pass an address and distance (bounding box or network) in meters¶

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:

network from address, including only nodes within 1km along the network from the address

G = ox.graph_from_address(

```
    address="Rruga Skënderbeu, Liri Gero, Sheq i Vogël, Fier, Bashkia Fier, Fier County, Southern Albania, 9301 - 9305, Albania",
    dist=250,
    dist_type="network",
    network_type="drive",
)


# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

Method #5, pass a place name¶

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Fier, Fier, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Fier, Fier, Albania",
    {"city": "Fier", "state": "Albania"},
    "Fier, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [13]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

Method #6, pass a polygon¶

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:

```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")

# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]

# polygon = mission_district["geometry"].iloc[0]


# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

Method #7, load a .osm xml file¶

In [15]:

```
# # create graph from .osm extract file

# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

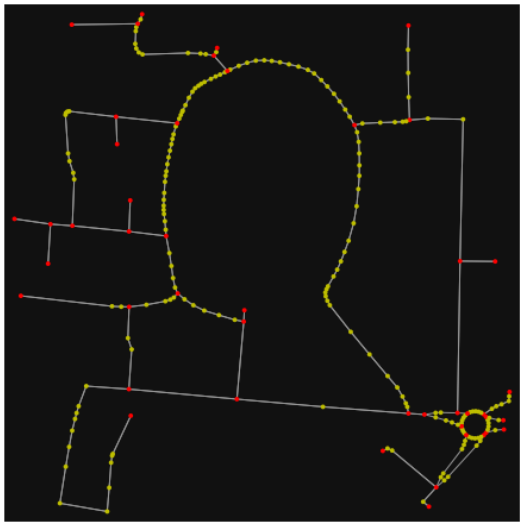
In [16]:

```
# create a network around some (lat, lng) point but do not simplify it yet

location_point = (40.7355, 19.5684)
```

```
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250,
simplify=False)
```

In [17]:

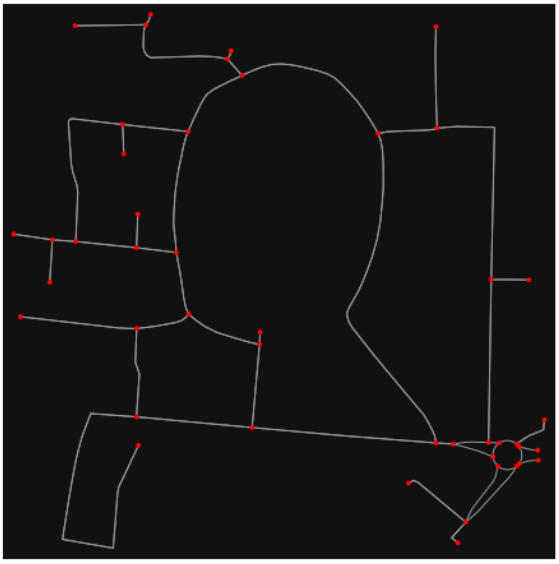
```
# turn off strict mode and see what nodes we'd remove, in yellow

nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]

fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [18]:

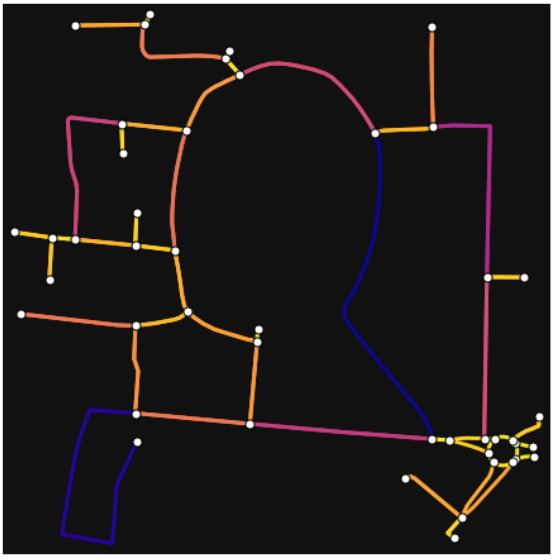
```
# simplify the network

G = ox.simplify_graph(G)

fig, ax = ox.plot_graph(G, node_color="r")
```



In [19]:

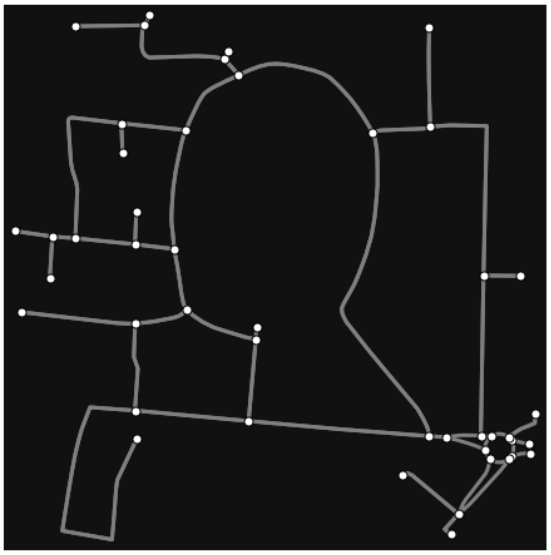
```
# show the simplified network with edges colored by length
```

```
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")

fig, ax = ox.plot_graph(

    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec,
edge_linewidth=3

)
```



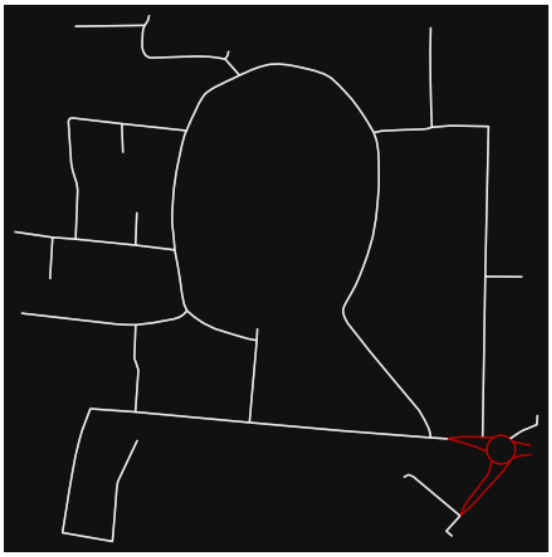
In [20]:

```
# highlight all parallel (multiple) edges

ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]

fig, ax = ox.plot_graph(

    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec,
edge_linewidth=3

)
```



In [21]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5,
edge_alpha=0.7)
```



Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:

```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/comunists_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/comunists_zone_network.graphml")
```

Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

2.466666666666667

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:

```
# calculate basic street network metrics and display average circuity
```

```
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

37

In [26]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[26]:

178802.42043799875

In [27]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[27]:

```
{'n': 45,
 'm': 88,
 'k_avg': 3.911111111111111,
 'edge_length_total': 6416.467,
 'edge_length_avg': 72.91439772727273,
 'streets_per_node_avg': 2.466666666666667,
 'streets_per_node_counts': {0: 0, 1: 8, 2: 9, 3: 27, 4: 1},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.17777777777777778,
  2: 0.2,
  3: 0.6,
  4: 0.022222222222222223},
 'intersection_count': 37,
 'street_length_total': 3372.935,
 'street_segment_count': 51,
 'street_length_avg': 66.13598039215687,
 'circuity_avg': 1.1608399457079759,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 21,
 'node_density_km': 251.674445400498,
```

'intersection_density_km': 206.93232177374279,

'edge_density_km': 35885.79497012438,

'street_density_km': 18864.034344376192,

'clean_intersection_density_km': 117.4480745202324}

In [28]:

```
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	45
m	88
k_avg	3.911111
edge_length_total	6416.467
edge_length_avg	72.914398
streets_per_node_avg	2.466667
streets_per_node_counts	{0: 0, 1: 8, 2: 9, 3: 27, 4: 1}
streets_per_node_proportions	{0: 0.0, 1: 0.1777777777777778, 2: 0.2, 3: 0....
intersection_count	37

	value
street_length_total	3372.935
street_segment_count	51
street_length_avg	66.13598
circuity_avg	1.161222
self_loop_proportion	0.0
node_density_km	251.674445
intersection_density_km	206.932322
edge_density_km	35885.79497
street_density_km	18864.034344
0way_int_count	0
1way_int_count	8
2way_int_count	9
3way_int_count	27
4way_int_count	1
0way_int_prop	0.0
1way_int_prop	0.177778
2way_int_prop	0.2
3way_int_prop	0.6
4way_int_prop	0.022222

In [30]:

```
import networkx as nx
```

In [31]:

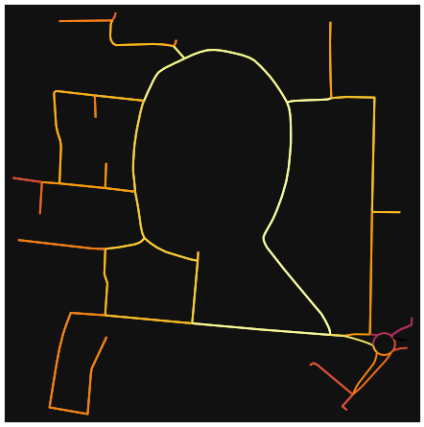
```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
```

```
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```



In [33]:

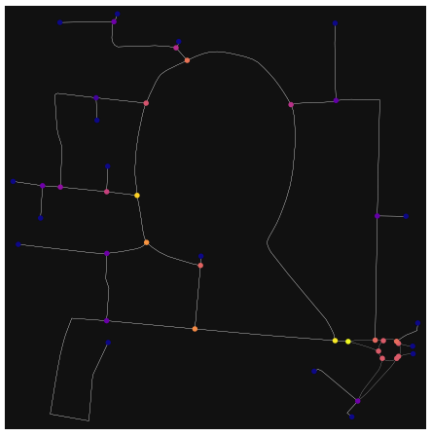
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

```
(6518905247, 0.4117336152219873)
```

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 16: KAMPIONI URBAN FIER 5

In [1]:

```
import geopandas as gpd

import osmnx as ox


%matplotlib inline

ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111:
UserWarning: The Shapely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the
GEOS version PyGEOS was compiled with (3.10.1-CAPI-1.16.0). Conversions between both
will be slow.

  warnings.warn(
```

Out[1]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off

ox.settings.use_cache = False


# turn it back on and turn on/off logging to your console

ox.settings.use_cache = True

ox.settings.log_console = False
```

Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:

```
# get the boundary polygon for Fier, project it, and plot it

city = ox.geocode_to_gdf("Fier, Fier, Albania")

city_proj = ox.project_gdf(city)

ax = city_proj.plot(fc="gray", ec="none")

_ = ax.axis("off")
```

In [4]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot

place_names = [

    "Fier, Fier, Albania",

    "Korçë, Korçë County, Albania",

    "Tiranë, Tirana County, Albania",

  ]

albania_place = ox.geocode_to_gdf(place_names)

albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")

albania_place = ox.project_gdf(albania_place)

ax = albania_place.plot(fc="gray", ec="none")

_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning:
pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use
pandas.Index with the appropriate dtype instead.

  pd.Int64Index,
```

In [5]:

if you know the OSM ID of the place(s) you want, you can query it directly

ox.geocode_to_gdf(["W706659044"], by_osmid=True)

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
(	LINESTRING (19.55730 40.73654, 19.55729 40.736...	40.738321	40.736535	19.55732	19.55648	241329005	way	706659044	40.737632	19.55728	Rruga Gjon Banushi, Mbrostar, Fier, Bashkia Fi...	highway	residential	0.1

Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

a bounding box

a lat-long point plus a distance

an address plus a distance

a place name or list of place names (to automatically geocode and get the boundary of)

a polygon of the desired street network's boundaries

a .osm formatted xml file

You can also specify several different network types:

'drive' - get drivable public streets (but not service roads)

'drive_service' - get drivable streets, including service roads

'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)

'bike' - get all streets and paths that cyclists can use

'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)

'all_private' - download all OSM streets and paths, including private-access ones

Method #1, pass a bounding box¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:

```
# define a bounding box in Fieri

north, south, east, west = 40.7407, 40.7357, 19.5548, 19.5598


# create network from that bounding box

G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

Method #2, pass a lat-lng point and bounding box distance in meters¶

This creates a bounding box n meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:

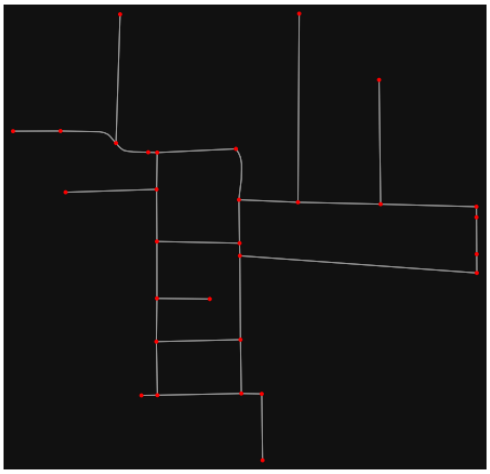
```
# define a point at the corner of the interested zone

location_point = (40.73798, 19.55728)


# create network from point, inside bounding box of N, S, E, W each 1000m from point

G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

Method #3, pass a lat-lng point and network distance in meters¶

This creates a bounding box n meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than n meters from the point along the network.

In [8]:

```
# same point again, but create network only of nodes within 250m along the network from point

G = ox.graph_from_point(location_point, dist=250, dist_type="network")

fig, ax = ox.plot_graph(G, node_color="r")
```



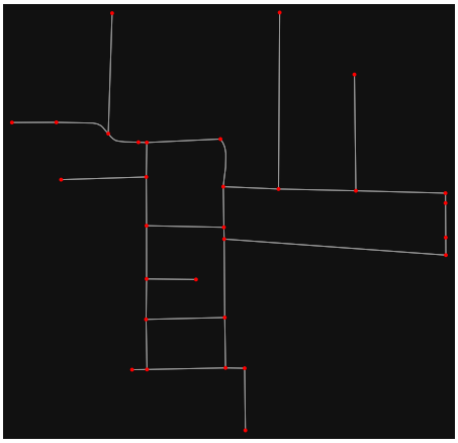
Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

create network only of nodes within 250m walking along the network from point

G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")

fig, ax = ox.plot_graph(G, node_color="r")



Method #4, pass an address and distance (bounding box or network) in meters¶

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:

network from address, including only nodes within 1km along the network from the address

```
G = ox.graph_from_address(
    address="Rruga Gjon Banushi, Mbrostar, Fier, Bashkia Fier, Fier County, Southern Albania, 9301, Albania",
    dist=250,
    dist_type="network",
    network_type="drive",
)
# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

Method #5, pass a place name¶

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Fier, Fier, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Fier, Fier, Albania",
    {"city": "Fier", "state": "Albania"},
    "Fier, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [13]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

Method #6, pass a polygon¶

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:

```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")

# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]

# polygon = mission_district["geometry"].iloc[0]


# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

Method #7, load a .osm xml file¶

In [15]:

```
# # create graph from .osm extract file

# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

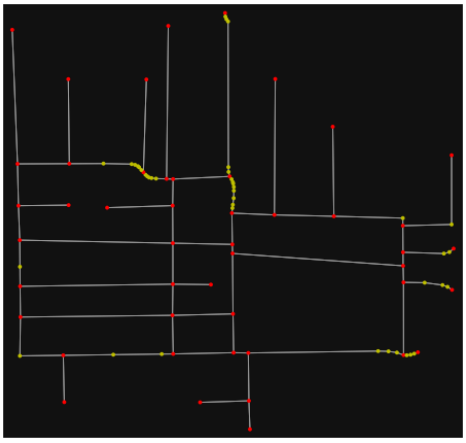
In [16]:

```
# create a network around some (lat, lng) point but do not simplify it yet

location_point = (40.73798, 19.55728)
```

```
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250,
simplify=False)
```

In [17]:

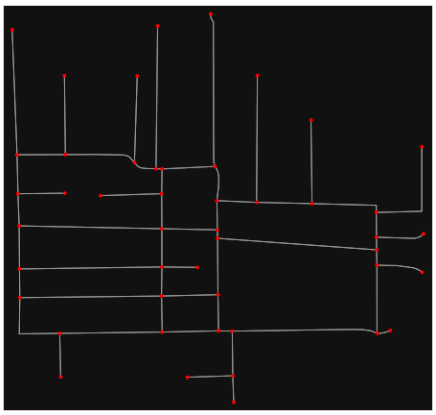
```
# turn off strict mode and see what nodes we'd remove, in yellow

nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]

fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [18]:

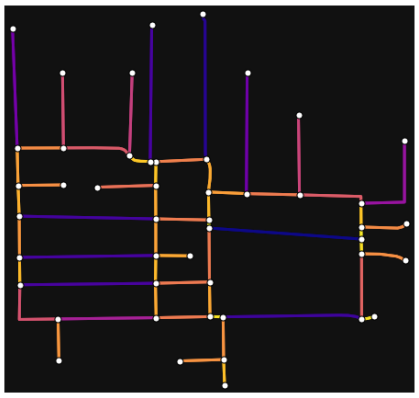
```
# simplify the network

G = ox.simplify_graph(G)

fig, ax = ox.plot_graph(G, node_color="r")
```



In [19]:

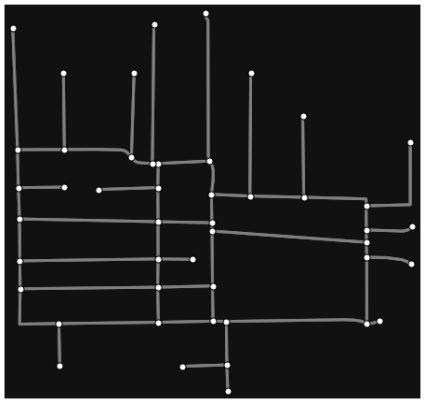
```
# show the simplified network with edges colored by length

ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")

fig, ax = ox.plot_graph(

    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec,
edge_linewidth=3
```

)



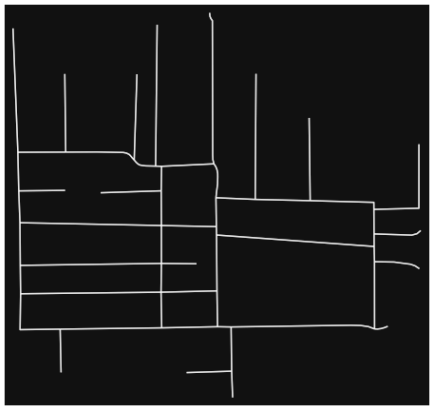
In [20]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec,
edge_linewidth=3
)
```



In [21]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5,
edge_alpha=0.7)
```



Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:

```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/comunists_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/comunists_zone_network.graphml")
```

Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

2.5319148936170213

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

36

In [26]:

```
G_proj = ox.project_graph(G)
```

```
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
Out[26]:
187456.09789040775
In [27]:
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
Out[27]:
{'n': 47,
 'm': 110,
 'k_avg': 4.680851063829787,
 'edge_length_total': 8157.892000000003,
 'edge_length_avg': 74.16265454545457,
 'streets_per_node_avg': 2.5319148936170213,
 'streets_per_node_counts': {0: 0, 1: 11, 2: 5, 3: 26, 4: 5},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.23404255319148937,
  2: 0.10638297872340426,
  3: 0.5531914893617021,
  4: 0.10638297872340426},
 'intersection_count': 36,
 'street_length_total': 4078.9460000000004,
 'street_segment_count': 55,
 'street_length_avg': 74.16265454545456,
 'circuity_avg': 1.0199777116688475,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 27,
 'node_density_km': 250.72537265486855,
 'intersection_density_km': 192.0449662888355,
 'edge_density_km': 43518.94705911003,
 'street_density_km': 21759.473529555013,
 'clean_intersection_density_km': 144.0337247166266}
In [28]:
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	47
m	110
k_avg	4.680851
edge_length_total	8157.892
edge_length_avg	74.162655
streets_per_node_avg	2.531915
streets_per_node_counts	{0: 0, 1: 11, 2: 5, 3: 26, 4: 5}
streets_per_node_proportions	{0: 0.0, 1: 0.23404255319148937, 2: 0.10638297...
intersection_count	36
street_length_total	4078.946
street_segment_count	55
street_length_avg	74.162655
circuity_avg	1.020467
self_loop_proportion	0.0

	value
node_density_km	250.725373
intersection_density_km	192.044966
edge_density_km	43518.947059
street_density_km	21759.47353
0way_int_count	0
1way_int_count	11
2way_int_count	5
3way_int_count	26
4way_int_count	5
0way_int_prop	0.0
1way_int_prop	0.234043
2way_int_prop	0.106383
3way_int_prop	0.553191
4way_int_prop	0.106383

In [30]:

```
import networkx as nx
```

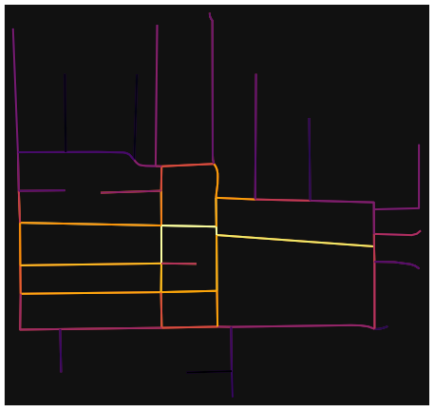
In [31]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```



In [33]:

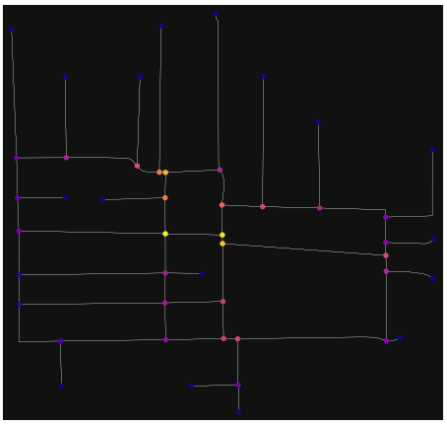
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

(1826149304, 0.3497584541062802)

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 17: KAMPIONI URBAN KORCA 1

In [39]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

Out[39]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [40]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

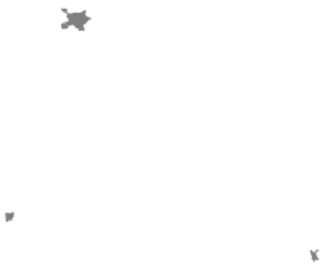
- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [41]:
```
# get the boundary polygon for Korce, project it, and plot it
city = ox.geocode_to_gdf("Korçë, Korçë County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [42]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```



In [43]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W683557752"], by_osmid=True)
```

Out[43]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (20.77717 40.61600, 20.77712	40.61684	40.61601	20.777168	20.77760	236968668	way	683557752	40.616452	20.77661	Bulevardi Fan Noli, Korçë, Bashkia Korç	highway	secondary	0.1

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
	40.616...										ë, Korç...			

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box*¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [44]:
```
# define a bounding box in Korca
north, south, east, west = 40.6203200, 40.6126200, 20.7843400, 20.7700000

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

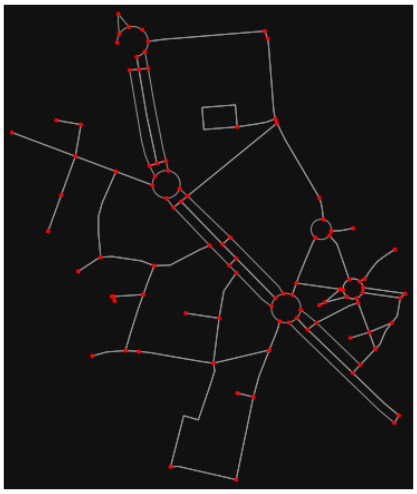
In [45]:
```
# define a point at the corner of the interested zone
location_point = (40.6164,20.7766)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters*¶

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [46]:

```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```



Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [47]:

```
# create network only of nodes within 1000m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```

- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [48]:
```
# network from address, including only nodes within 500m along the network from the address
G = ox.graph_from_address(
    address="Bulevardi Fan Noli, Korçë, Bashkia Korçë, Korçë County, Southern Albania, 7001, Albani
a",
    dist=250,
    dist_type="network",
    network_type="drive",
)

# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [49]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Korçë, Korçë County, Albania", network_type="drive")
```

In [50]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Korçë, Korçë County, Albania",
    {"city": "Korçë", "state": "Albania"},
    "Korçë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [51]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [52]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [53]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

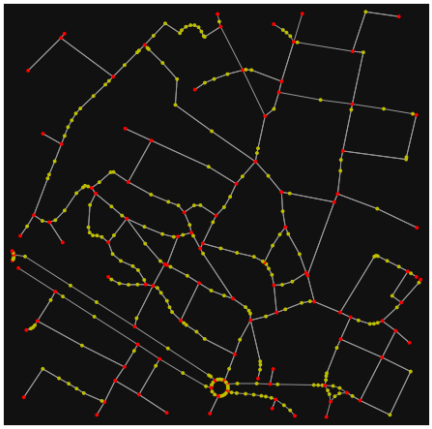
- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [54]:
```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.61501, 20.78229)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

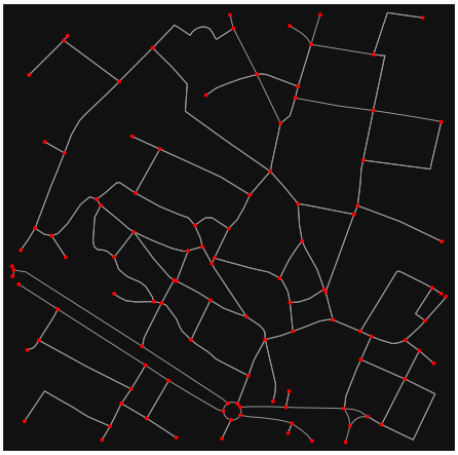
In [55]:

```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



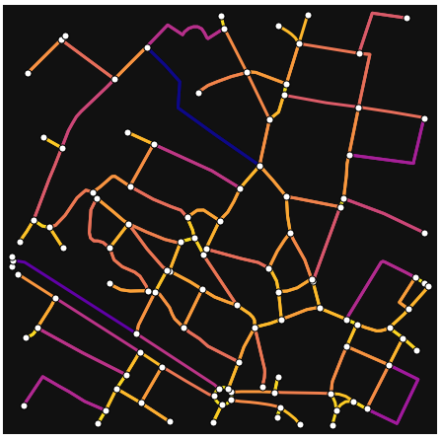
The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [56]:
```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



In [57]:

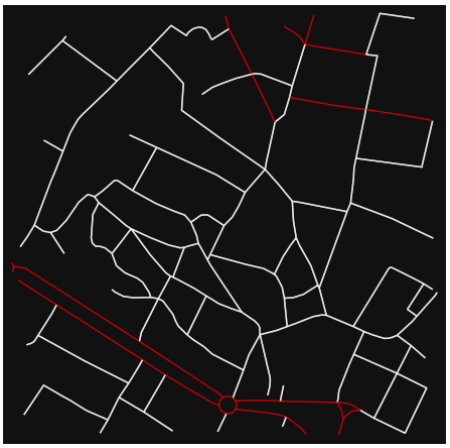
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [58]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [59]:
```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [60]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/korca_comunists_zone_network.gpkg")
```

In [61]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/korca_comunists_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [62]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[62]:

2.814814814814815

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [63]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[63]:

99

In [64]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
```

```
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[64]:

```
225418.45950680517
```

In [65]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[65]:

```
{'n': 108,
 'm': 244,
 'k_avg': 4.518518518518518,
 'edge_length_total': 12234.606999999987,
 'edge_length_avg': 50.141831967213065,
 'streets_per_node_avg': 2.814814814814815,
 'streets_per_node_counts': {0: 0, 1: 9, 2: 15, 3: 71, 4: 13},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.08333333333333333,
  2: 0.1388888888888889,
  3: 0.6574074074074074,
  4: 0.12037037037037036},
 'intersection_count': 99,
 'street_length_total': 6815.9,
 'street_segment_count': 137,
 'street_length_avg': 49.75109489051095,
 'circuity_avg': 1.0493646458091637,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 58,
 'node_density_km': 479.10894359004163,
 'intersection_density_km': 439.1831982908715,
 'edge_density_km': 54275.08921304928,
 'street_density_km': 30236.654153845968,
 'clean_intersection_density_km': 257.29924748354085}
```

In [66]:

```
import pandas as pd
```

In [67]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[67]:

	value
n	108
m	244
k_avg	4.518519
edge_length_total	12234.607
edge_length_avg	50.141832
streets_per_node_avg	2.814815
streets_per_node_counts	{0: 0, 1: 9, 2: 15, 3: 71, 4: 13}
streets_per_node_proportions	{0: 0.0, 1: 0.08333333333333333, 2: 0.13888888...
intersection_count	99
street_length_total	6815.9
street_segment_count	137
street_length_avg	49.751095
circuity_avg	1.049724
self_loop_proportion	0.0
node_density_km	479.108944
intersection_density_km	439.183198
edge_density_km	54275.089213
street_density_km	30236.654154
0way_int_count	0
1way_int_count	9
2way_int_count	15
3way_int_count	71
4way_int_count	13
0way_int_prop	0.0
1way_int_prop	0.083333
2way_int_prop	0.138889
3way_int_prop	0.657407
4way_int_prop	0.12037

In [68]:

```
import networkx as nx
```

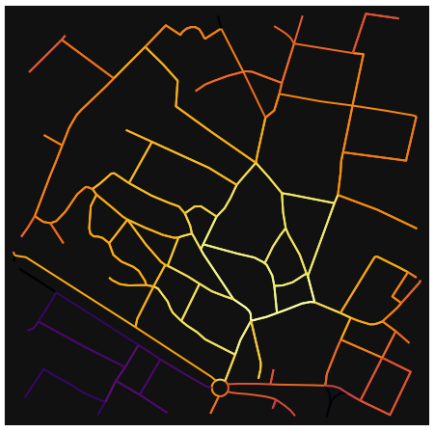
In [69]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [70]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```



In [71]:

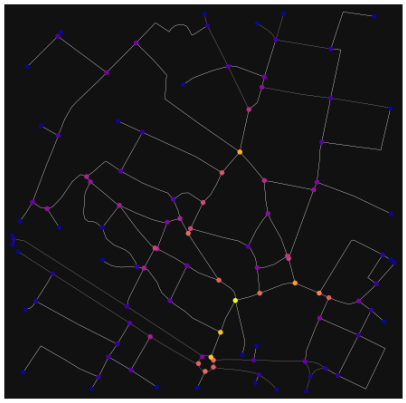
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[71]:

(711507008, 0.2609769000176336)

In [72]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



In [ ]:

ANEKSI 18: KAMPIONI URBAN KORCA 2

In [1]:

```
import geopandas as gpd
import osmnx as ox
```

```
%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

```
'1.2.0'
```

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Korce, project it, and plot it
city = ox.geocode_to_gdf("Korçë, Korçë County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
   ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
pd.Int64Index,

In [5]:
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W57009658"], by_osmid=True)

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (20.77165 40.61820, 20.77263 40.617...	40.61821	40.61599	20.77454	20.77146	116470909	way	57009658	40.617019	20.773016	Rruga Ajet Xhindolli, Korçë, Bashkia Korçë, Ko...	highway	residential	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use

- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Korca 2nd Zone
north, south, east, west = 40.6193, 40.6140, 20.7759, 20.7709

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

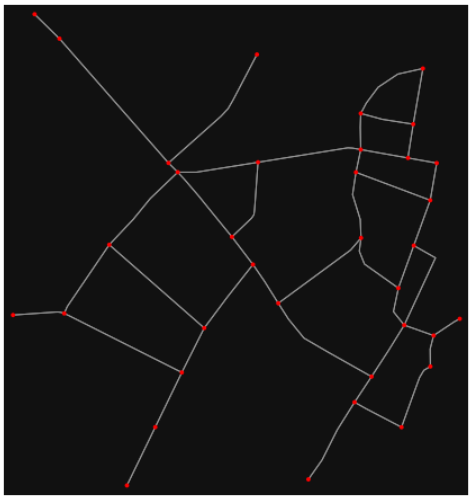
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (40.61699, 20.77304)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

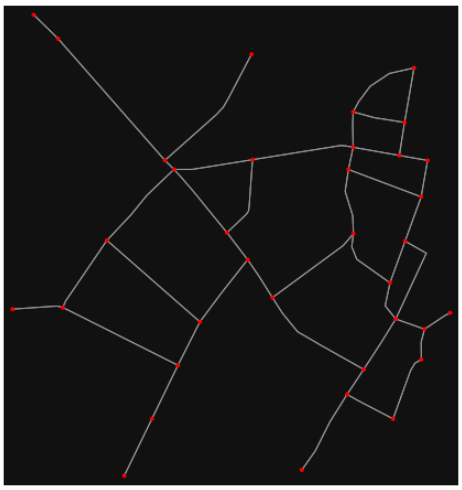


Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is

allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

```
# create network only of nodes within 500m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:

```
# network from address, including only nodes within 500m along the network from the address
G = ox.graph_from_address(
    address="Rruga Ajet Xhindolli, Korçë, Bashkia Korçë, Korçë County, Southern Albania, 7001, Alba
nia",
    dist=250,
    dist_type="network",
    network_type="drive",
)

# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Korçë, Korçë County, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Korçë, Korçë County, Albania",
    {"city": "Korçë", "state": "Albania"},
    "Korçë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [13]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:

```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [15]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

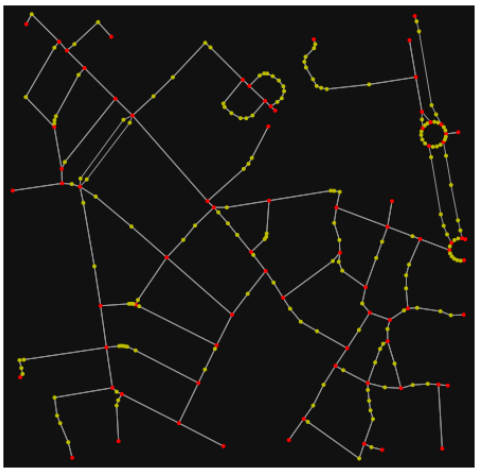
Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [16]:

```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.61699, 20.77304)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [17]:

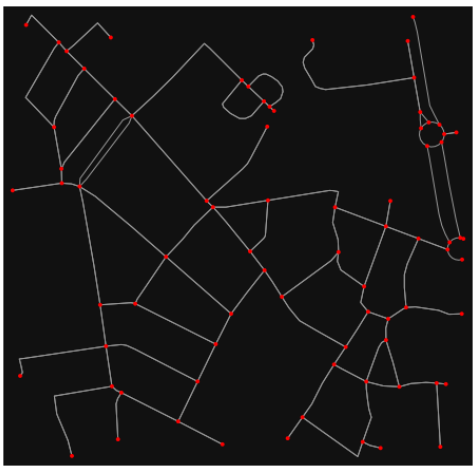
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

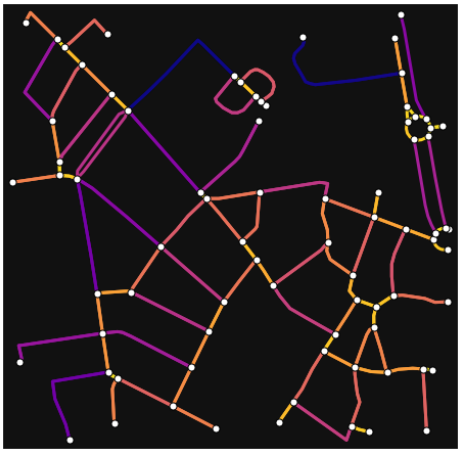
In [18]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



In [19]:

```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [20]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [21]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```

- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/korca_2nd_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/korca_2nd_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

2.8181818181818183

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:
```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

73

In [26]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[26]:

226526.37856456143

In [27]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[27]:

{'n': 77,
'm': 181,
'k_avg': 4.701298701298701,
'edge_length_total': 10279.400000000005,
'edge_length_avg': 51.79226519337019,
'streets_per_node_avg': 2.8181818181818183,
'streets_per_node_counts': {0: 0, 1: 4, 2: 16, 3: 49, 4: 6, 5: 2},
'streets_per_node_proportions': {0: 0.0,
1: 0.05194805194805195,
2: 0.2077922077922078,

```
 3: 0.6363636363636364,
 4: 0.07792207792207792,
 5: 0.025974025974025976},
'intersection_count': 73,
'street_length_total': 5493.136000000001,
'street_segment_count': 99,
'street_length_avg': 50.48622222222224,
'circuity_avg': 1.0781504109937723,
'self_loop_proportion': 0.0,
'clean_intersection_count': 43,
'node_density_km': 339.916262679556,
'intersection_density_km': 322.25827500789075,
'edge_density_km': 45378.37961802896,
'street_density_km': 24249.43194169514,
'clean_intersection_density_km': 189.8233674704014}
```

In [28]:

```
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	77
m	181
k_avg	4.701299
edge_length_total	10279.4
edge_length_avg	51.792265
streets_per_node_avg	2.818182
streets_per_node_counts	{0: 0, 1: 4, 2: 16, 3: 49, 4: 6, 5: 2}
streets_per_node_proportions	{0: 0.0, 1: 0.05194805194805195, 2: 0.20779220...
intersection_count	73
street_length_total	5493.136
street_segment_count	99
street_length_avg	50.486222
circuity_avg	1.078167
self_loop_proportion	0.0
node_density_km	339.916263
intersection_density_km	322.258275

	value
edge_density_km	45378.379618
street_density_km	24249.431942
0way_int_count	0
1way_int_count	4
2way_int_count	16
3way_int_count	49
4way_int_count	6
5way_int_count	2
0way_int_prop	0.0
1way_int_prop	0.051948
2way_int_prop	0.207792
3way_int_prop	0.636364
4way_int_prop	0.077922
5way_int_prop	0.025974

In [30]:

```
import networkx as nx
```

In [31]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [33]:

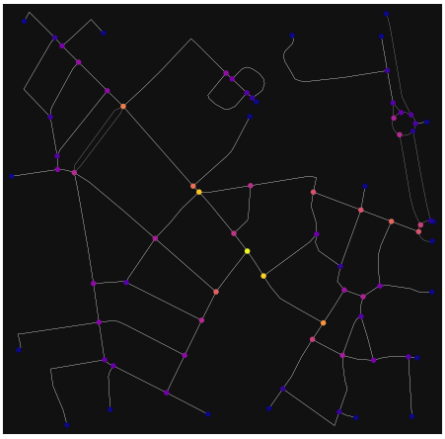
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

(2627199486, 0.29175438596491227)

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 19: KAMPIONI URBAN KORCA 3

In [1]:

```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas_compat.py:111: UserWarning: The Shapely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compiled with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(

Out[1]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
```

```
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Korce, project it, and plot it
city = ox.geocode_to_gdf("Korçë, Korçë County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
   ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```

In [5]:
```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W194537121"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (20.77934 40.61137, 20.77972 40.611...	40.61247	40.61365	20.78027	20.77934	147803299	way	19453712	40.61176	20.77971	Rruga Jorgjia Lubonja, Korçë, Bashkia Korçë, K...	highway	residential	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box*¶

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Korca 3rd Zone
north, south, east, west = 40.6144, 40.6091, 20.7771, 20.7821

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters*¶

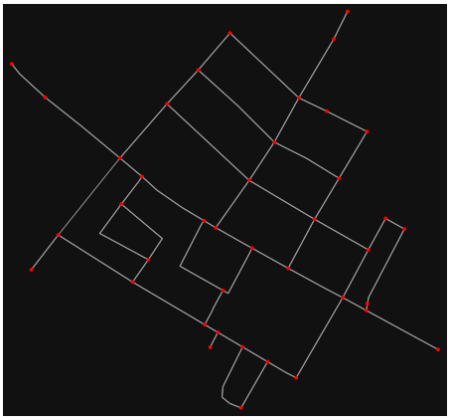
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (40.61137, 20.77935)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters*¶

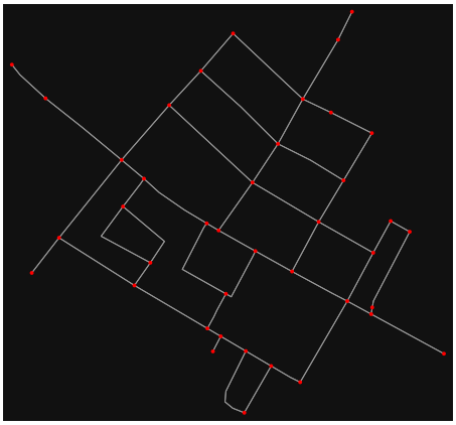
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```



Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:
```
# create network only of nodes within 500m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:
```
# network from address, including only nodes within 500m along the network from the address
G = ox.graph_from_address(
    address="Rruga Jorgjia Lubonja, Korçë, Bashkia Korçë, Korçë County, Southern Albania, 7001, Alb
ania",
    dist=250,
    dist_type="network",
    network_type="drive",
)

# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:
```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Korçë, Korçë County, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Korçë, Korçë County, Albania",
    {"city": "Korçë", "state": "Albania"},
    "Korçë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [13]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:
```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [15]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

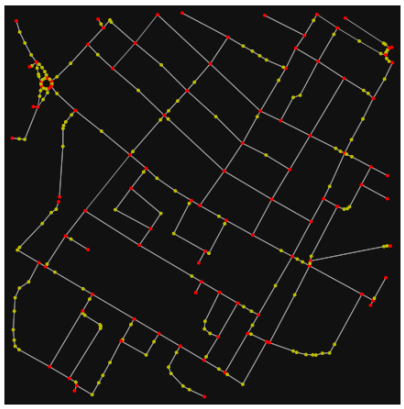
- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [16]:
```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.61137, 20.77935)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

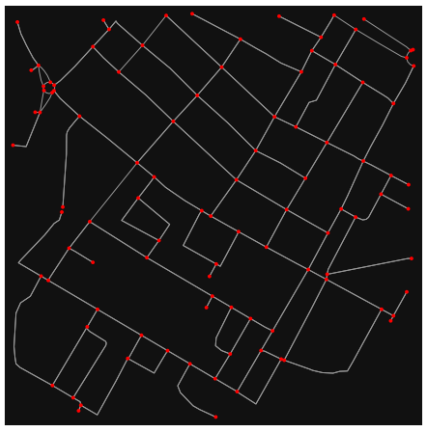
In [17]:

```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

In [18]:
```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



In [19]:

```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [20]:

```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```

In [21]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```

- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:

```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/korca_3rd_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/korca_3rd_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

2.9603960396039604

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

96

In [26]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[26]:

215602.26286305822

In [27]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[27]:

```
{'n': 101,
 'm': 255,
 'k_avg': 5.0495049504950495,
 'edge_length_total': 13456.504000000012,
 'edge_length_avg': 52.77060392156867,
 'streets_per_node_avg': 2.9603960396039604,
 'streets_per_node_counts': {0: 0, 1: 5, 2: 13, 3: 64, 4: 19},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.04950495049504951,
  2: 0.12871287128712872,
  3: 0.6336633663366337,
  4: 0.18811881188118812},
 'intersection_count': 96,
 'street_length_total': 6998.223999999999,
 'street_segment_count': 137,
 'street_length_avg': 51.081927007299264,
 'circuity_avg': 1.0518103900763822,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 57,
 'node_density_km': 468.45519457349616,
 'intersection_density_km': 445.26434335698644,
 'edge_density_km': 62413.556431673605,
 'street_density_km': 32458.954312761485,
 'clean_intersection_density_km': 264.3757038682107}
```

In [28]:

```
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	101
m	255
k_avg	5.049505

	value
edge_length_total	13456.504
edge_length_avg	52.770604
streets_per_node_avg	2.960396
streets_per_node_counts	{0: 0, 1: 5, 2: 13, 3: 64, 4: 19}
streets_per_node_proportions	{0: 0.0, 1: 0.04950495049504951, 2: 0.12871287...
intersection_count	96
street_length_total	6998.224
street_segment_count	137
street_length_avg	51.081927
circuity_avg	1.0519
self_loop_proportion	0.0
node_density_km	468.455195
intersection_density_km	445.264343
edge_density_km	62413.556432
street_density_km	32458.954313
0way_int_count	0
1way_int_count	5
2way_int_count	13
3way_int_count	64
4way_int_count	19
0way_int_prop	0.0
1way_int_prop	0.049505
2way_int_prop	0.128713
3way_int_prop	0.633663
4way_int_prop	0.188119

In [30]:

```
import networkx as nx
```

In [31]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [33]:

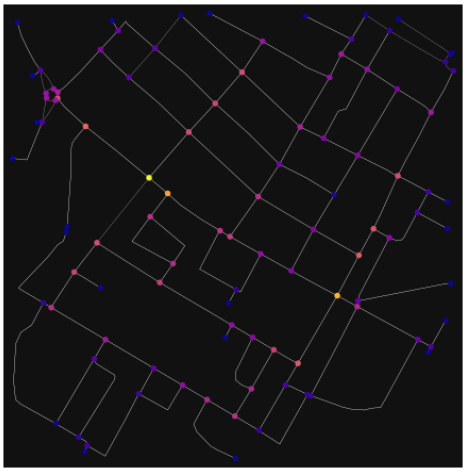
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

(2050375568, 0.282020202020202)

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



In [ ]:

ANEKSI 20: KAMPIONI URBAN KORCA 4

In [1]:

```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

```
'1.2.0'
```

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:

```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:

```
# get the boundary polygon for Korce, project it, and plot it
city = ox.geocode_to_gdf("Korçë, Korçë County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:

```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
```

```
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```

In [5]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W192575650"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (20.78296 40.62115, 20.78220 40.621...	40.62469	40.62146	20.78296	20.77914	146999867	way	192575650	40.62148	20.78160	Rruga Vojo Kushi, Korçë, Bashkia Korçë, Korçë ...	highway	residential	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads

- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Korca 4rth Zone
north, south, east, west = 40.6245, 40.6192, 20.7789, 20.7839

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

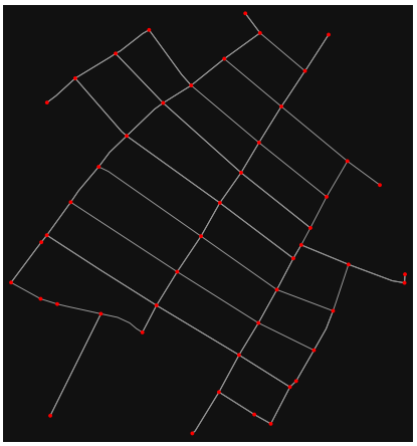
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (40.62191, 20.7808)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

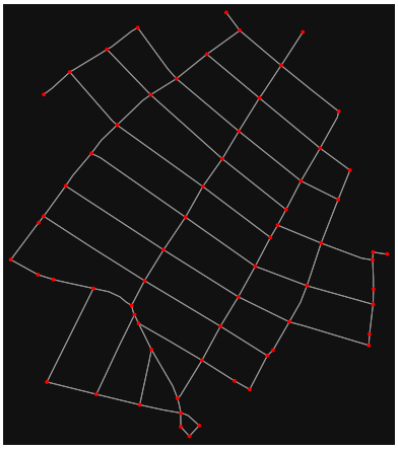


Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead

(below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

```
# create network only of nodes within 500m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:

```
# network from address, including only nodes within 500m along the network from the address
G = ox.graph_from_address(
    address="Rruga Vojo Kushi, Korçë, Bashkia Korçë, Korçë County, Southern Albania, 7001, Albania
",
    dist=250,
    dist_type="network",
    network_type="drive",
)

# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Korçë, Korçë County, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Korçë, Korçë County, Albania",
    {"city": "Korçë", "state": "Albania"},
    "Korçë, Albania",
]
G = ox.graph_from_place(places, truncate_by_edge=True)
```

In [13]:

```
# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)
```

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:

```
# calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
# mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
# polygon = mission_district["geometry"].iloc[0]

# G2 = ox.graph_from_polygon(polygon, network_type="drive_service")
```

- *Method #7, load a .osm xml file¶*

In [15]:

```
# # create graph from .osm extract file
# G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")
```

- Part 3: simplifying street network topology¶

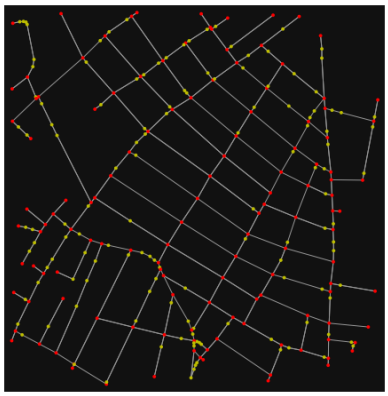
Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [16]:

```
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.62191, 20.7808)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)
```

In [17]:

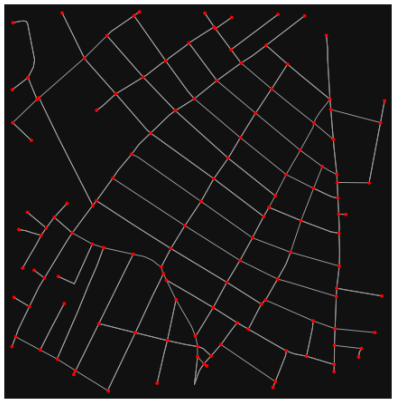
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

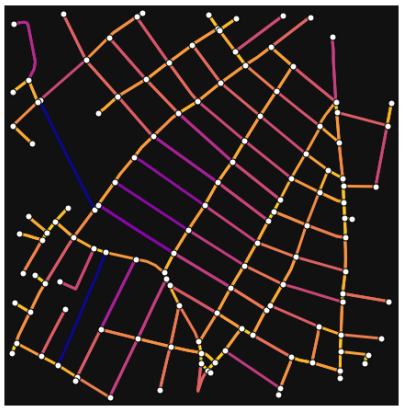
In [18]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```



In [19]:

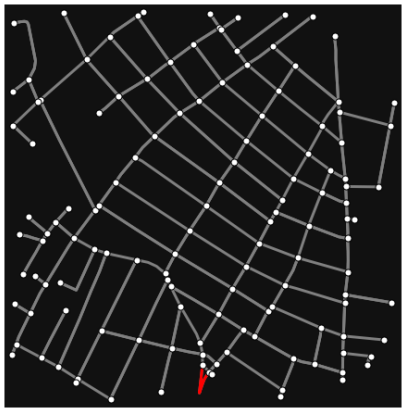
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [20]:

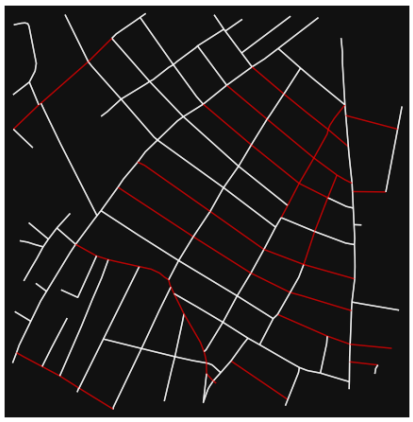
```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
```

```
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [21]:

```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:

```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/korca_4th_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/korca_4th_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

3.0615384615384613

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

122

In [26]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[26]:

227880.92603160645

In [27]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[27]:

```
{'n': 130,
 'm': 309,
 'k_avg': 4.753846153846154,
 'edge_length_total': 13773.583999999983,
 'edge_length_avg': 44.57470550161807,
 'streets_per_node_avg': 3.0615384615384613,
 'streets_per_node_counts': {0: 0, 1: 8, 2: 19, 3: 60, 4: 43},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.06153846153846154,
  2: 0.14615384615384616,
  3: 0.46153846153846156,
  4: 0.33076923076923076},
 'intersection_count': 122,
 'street_length_total': 8116.138999999998,
 'street_segment_count': 181,
 'street_length_avg': 44.840546961325956,
 'circuity_avg': 1.0520042619410041,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': 73,
 'node_density_km': 570.4733707373532,
 'intersection_density_km': 535.3673171535162,
 'edge_density_km': 60442.02224318513,
 'street_density_km': 35615.701328483774,
 'clean_intersection_density_km': 320.34273895251374}
```

In [28]:

```
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
```

```
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	130
m	309
k_avg	4.753846
edge_length_total	13773.584
edge_length_avg	44.574706
streets_per_node_avg	3.061538
streets_per_node_counts	{0: 0, 1: 8, 2: 19, 3: 60, 4: 43}
streets_per_node_proportions	{0: 0.0, 1: 0.06153846153846154, 2: 0.14615384...
intersection_count	122
street_length_total	8116.139
street_segment_count	181
street_length_avg	44.840547
circuity_avg	1.052135
self_loop_proportion	0.0
node_density_km	570.473371
intersection_density_km	535.367317
edge_density_km	60442.022243
street_density_km	35615.701328
0way_int_count	0
1way_int_count	8
2way_int_count	19
3way_int_count	60
4way_int_count	43
0way_int_prop	0.0
1way_int_prop	0.061538
2way_int_prop	0.146154
3way_int_prop	0.461538
4way_int_prop	0.330769

In [30]:

```
import networkx as nx
```

In [31]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```

In [33]:

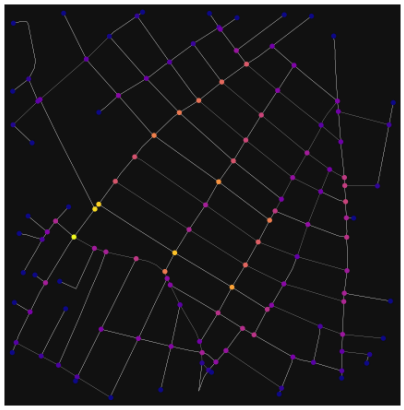
```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

(336168511, 0.21935562015503876)

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",
)
```



ANEKSI 21: KAMPIONI URBAN KORCA 5

In [1]:
```
import geopandas as gpd
import osmnx as ox

%matplotlib inline
ox.__version__
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\_compat.py:111: UserWarning: The Sh
apely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS version PyGEOS was compi
led with (3.10.1-CAPI-1.16.0). Conversions between both will be slow.
  warnings.warn(
```

Out[1]:

'1.2.0'

You can configure OSMnx using the settings module. See the documentation for the settings you can configure. For example, by default OSMnx caches all server responses to prevent repeatedly hitting the server for the same query every time you run it. This both makes our code faster on subsequent runs and helps us be a "good neighbor" to the server. But you can turn caching off (or back on again) with the use_cache setting.

In [2]:
```
# turn response caching off
ox.settings.use_cache = False

# turn it back on and turn on/off logging to your console
ox.settings.use_cache = True
ox.settings.log_console = False
```

- Part 1: get place boundaries from OpenStreetMap¶

OSMnx lets you download place boundary geometries from OpenStreetMap, project them, and plot them. For a more in-depth demonstration of querying by place, see this notebook.

In [3]:
```
# get the boundary polygon for Korce, project it, and plot it
city = ox.geocode_to_gdf("Korçë, Korçë County, Albania")
city_proj = ox.project_gdf(city)
ax = city_proj.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

In [4]:
```
# get boundary polygons for several cities, save as GeoPackage, project to UTM, and plot
place_names = [
```

```
    "Fier, Fier, Albania",
    "Korçë, Korçë County, Albania",
    "Tiranë, Tirana County, Albania",
  ]
albania_place = ox.geocode_to_gdf(place_names)
albania_place.to_file("./data/albania_place.gpkg", driver="GPKG")
albania_place = ox.project_gdf(albania_place)
ax = albania_place.plot(fc="gray", ec="none")
_ = ax.axis("off")
```

```
C:\Users\User\anaconda3\envs\ox\lib\site-packages\geopandas\io\file.py:362: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with th
e appropriate dtype instead.
  pd.Int64Index,
```

In [5]:

```
# if you know the OSM ID of the place(s) you want, you can query it directly
ox.geocode_to_gdf(["W461136812"], by_osmid=True)
```

Out[5]:

	geometry	bbox_north	bbox_south	bbox_east	bbox_west	place_id	osm_type	osm_id	lat	lon	display_name	class	type	importance
0	LINESTRING (20.77923 40.61466, 20.77907 40.6144...	40.6158172	40.6146057	20.779228	20.777351	203385014	way	461136812	40.615261	20.778211	Bulevardi Fan Noli, Korçë, Bashkia Korçë, Korçë ...	highway	secondary	0.1

- Part 2: download and model street networks¶

OSMnx lets you download street network data and build topologically-corrected street networks, project and plot the networks, and save the street network as SVGs, GraphML files, GeoPackages, or shapefiles for later use. The street networks are directed and preserve one-way directionality. For a more in-depth demonstration of creating street networks, see this notebook.

You can download a street network by providing OSMnx any of the following (demonstrated in the examples below):

- a bounding box
- a lat-long point plus a distance
- an address plus a distance
- a place name or list of place names (to automatically geocode and get the boundary of)
- a polygon of the desired street network's boundaries
- a .osm formatted xml file

You can also specify several different network types:

- 'drive' - get drivable public streets (but not service roads)
- 'drive_service' - get drivable streets, including service roads
- 'walk' - get all streets and paths that pedestrians can use (this network type ignores one-way directionality)
- 'bike' - get all streets and paths that cyclists can use
- 'all' - download all non-private OSM streets and paths (this is the default network type unless you specify a different one)
- 'all_private' - download all OSM streets and paths, including private-access ones

- *Method #1, pass a bounding box¶*

This constructs the network from all the OSM nodes and ways within the bounding box.

In [6]:
```
# define a bounding box in Korca 5th Zone
north, south, east, west = 40.6188, 40.6135, 20.7746, 20.7796

# create network from that bounding box
G = ox.graph_from_bbox(north, south, east, west, network_type="drive_service")
```

- *Method #2, pass a lat-lng point and bounding box distance in meters¶*

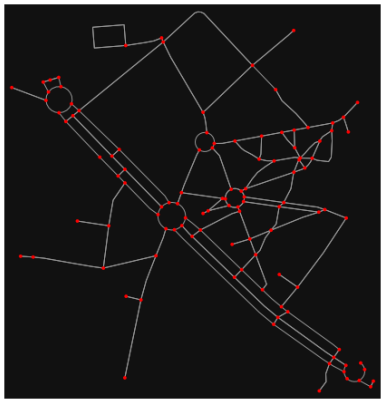
This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box.

In [7]:
```
# define a point at the corner of the interested zone
location_point = (40.61588, 20.77721)

# create network from point, inside bounding box of N, S, E, W each 250 m from point
G = ox.graph_from_point(location_point, dist=250, dist_type="bbox", network_type="drive")
```

- *Method #3, pass a lat-lng point and network distance in meters¶*

This creates a bounding box *n* meters North, South, East, and West of the point, then constructs the network from all the OSM nodes and ways within the bounding box. Then it truncates the network by removing all nodes further than *n* meters from the point along the network.

In [8]:
```
# same point again, but create network only of nodes within 250m along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network")
fig, ax = ox.plot_graph(G, node_color="r")
```

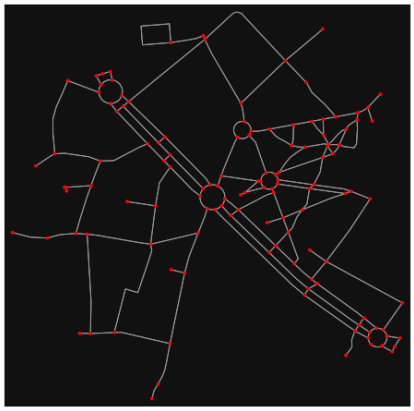


Note the plot above shows the network within 500m (traveling distance along the network) from the location_point. By default, the network_type parameter value is 'all', meaning that we do not filter out

paths that restrict certain types of traffic. This also means that one-way streets are honored as one-way and you cannot travel the wrong direction down them. Thus, the 500m takes into account only those nodes you can reach within 500m while only traveling in the allowed direction of the street. Instead (below), we can specify network_type='walk' to build a street network only of paths that walking is allowed on. This also makes every path bi-directional in the directed network, because you can walk in either direction on the sidewalk of a one-way street. Thus, the 500m now takes into account those nodes you can reach within 500m while traveling in either direction (even if it's a one-way street).

In [9]:

```
# create network only of nodes within 500m walking along the network from point
G = ox.graph_from_point(location_point, dist=250, dist_type="network", network_type="walk")
fig, ax = ox.plot_graph(G, node_color="r")
```



- *Method #4, pass an address and distance (bounding box or network) in meters¶*

This geocodes the address, creates a bounding box, downloads the network, then truncates it by network distance (if distance_type='network').

In [10]:

```
# network from address, including only nodes within 500m along the network from the address
G = ox.graph_from_address(
    address="Bulevardi Fan Noli, Korçë, Bashkia Korçë, Korçë County, Southern Albania, 7001, Albani
a",
    dist=250,
    dist_type="network",
    network_type="drive",
)

# you can project the network to UTM (zone calculated automatically)
G_projected = ox.project_graph(G)
```

- *Method #5, pass a place name¶*

This geocodes the place name, gets the place's boundary shape polygon and bounding box, downloads the network within the bounding box, then truncates it to the place's boundary polygon.

In [11]:

```
# create the street network within the city of Fieri's borders
G = ox.graph_from_place("Korçë, Korçë County, Albania", network_type="drive")
```

In [12]:

```
# you can also pass multiple places as a mixed list of strings and/or dicts
places = [
    "Korçë, Korçë County, Albania",
    {"city": "Korçë", "state": "Albania"},
    "Korçë, Albania",
```

]
G = ox.graph_from_place(places, truncate_by_edge=True)

In [13]:

# save to disk as GeoPackage file then plot
ox.save_graph_geopackage(G)
fig, ax = ox.plot_graph(G, node_size=0, edge_color="w", edge_linewidth=0.2)

- *Method #6, pass a polygon¶*

This example loads the Mission District's polygon from a shapefile, then downloads the network within its bounding box, then prunes all nodes that lie outside the place's boundary polygon.

In [14]:
calif = gpd.read_file("input_data/ZillowNeighborhoods-CA")
mission_district = calif[(calif["CITY"] == "San Francisco") & (calif["NAME"] == "Mission")]
polygon = mission_district["geometry"].iloc[0]

G2 = ox.graph_from_polygon(polygon, network_type="drive_service")

- *Method #7, load a .osm xml file¶*

In [15]:

create graph from .osm extract file
G = ox.graph_from_xml("./input_data/West-Oakland.osm.bz2")

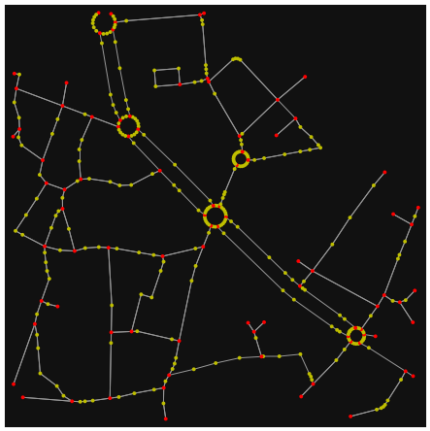
- Part 3: simplifying street network topology¶

Simplification is normally done by OSMnx automatically under the hood, but we can break it out to see how it works. OpenStreetMap nodes are weird. They include intersections, but they also include all the points along a single block where the street curves. The latter are not nodes in the graph theory sense, so we remove them algorithmically and consolidate the set of edges between "true" network nodes into a single edge. There are two simplification modes, strict and non-strict. The main difference is that unlike strict mode, non-strict mode allows simplification to an "expansion graph" (ie, if the graph were undirected, nodes with degree 2 as long as the incident edges have different OSM IDs). For a more in-depth demonstration of topological simplification with OSMnx, see this notebook.

In [16]:
# create a network around some (lat, lng) point but do not simplify it yet
location_point = (40.61588, 20.77721)
G = ox.graph_from_point(location_point, network_type="drive_service", dist=250, simplify=False)

In [17]:

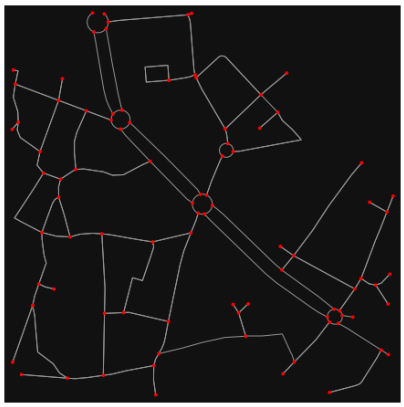
```
# turn off strict mode and see what nodes we'd remove, in yellow
nc = ["r" if ox.simplification._is_endpoint(G, node) else "y" for node in G.nodes()]
fig, ax = ox.plot_graph(G, node_color=nc)
```



The dots above are OSM nodes. We'll remove the nodes in yellow as they're not real network nodes (intersections/dead-ends).

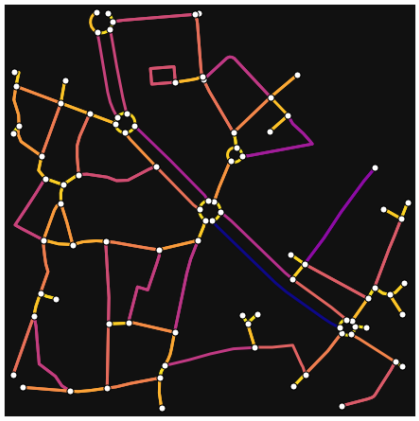
In [18]:

```
# simplify the network
G = ox.simplify_graph(G)
fig, ax = ox.plot_graph(G, node_color="r")
```

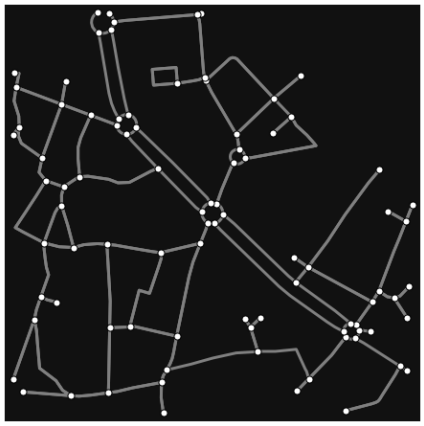


In [19]:

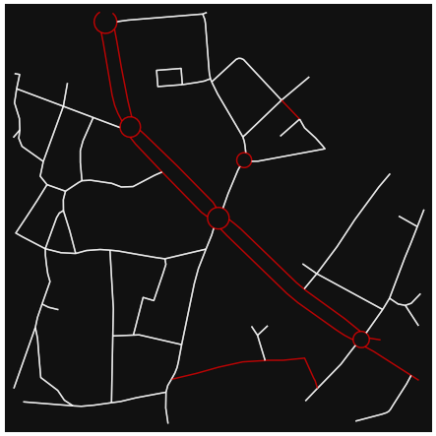
```
# show the simplified network with edges colored by length
ec = ox.plot.get_edge_colors_by_attr(G, attr="length", cmap="plasma_r")
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [20]:
```
# highlight all parallel (multiple) edges
ec = ["gray" if k == 0 or u == v else "r" for u, v, k in G.edges(keys=True)]
fig, ax = ox.plot_graph(
    G, node_color="w", node_edgecolor="k", node_size=50, edge_color=ec, edge_linewidth=3
)
```



In [21]:
```
# highlight all one-way edges in the mission district network from earlier
ec = ["r" if data["oneway"] else "w" for u, v, key, data in G.edges(keys=True, data=True)]
fig, ax = ox.plot_graph(G, node_size=0, edge_color=ec, edge_linewidth=1.5, edge_alpha=0.7)
```



- Part 4: saving networks to disk¶

For more examples of saving and loading networks to/from disk, see this notebook.

In [22]:
```
# save street network as GeoPackage to work with in GIS
ox.save_graph_geopackage(G, filepath="./data/korca_5th_zone_network.gpkg")
```

In [23]:

```
# save street network as GraphML file to work with later in OSMnx or networkx or gephi
ox.save_graphml(G, filepath="./data/korca_5th_zone_network.graphml")
```

- Part 5: calculate basic network indicators¶

In [24]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["streets_per_node_avg"]
```

Out[24]:

2.6818181818181817

In this street network, the streets are ~16% more circuitous than the straight-lines paths would be.

For examples of analyzing street networks, see this example.

In [25]:

```
# calculate basic street network metrics and display average circuity
stats = ox.basic_stats(G)
stats["intersection_count"]
```

Out[25]:

78

In [26]:

```
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[26]:

209263.39375248674

In [27]:

```
ox.basic_stats(G_proj, area=graph_area_m, clean_int_tol=15)
```

Out[27]:

```
{'n': 88,
 'm': 182,
 'k_avg': 4.136363636363637,
 'edge_length_total': 9269.972999999994,
 'edge_length_avg': 50.93391758241755,
 'streets_per_node_avg': 2.6818181818181817,
 'streets_per_node_counts': {0: 0, 1: 10, 2: 12, 3: 62, 4: 4},
 'streets_per_node_proportions': {0: 0.0,
  1: 0.11363636363636363,
  2: 0.13636363636363635,
  3: 0.7045454545454546,
  4: 0.045454545454545456},
 'intersection_count': 78,
 'street_length_total': 5422.304000000001,
 'street_segment_count': 110,
 'street_length_avg': 49.293672727272735,
 'circuity_avg': 1.0643573332754908,
 'self_loop_proportion': 0.00909090909090909,
 'clean_intersection_count': 36,
 'node_density_km': 420.52266486743946,
 'intersection_density_km': 372.7359984052304,
 'edge_density_km': 44298.11078646829,
 'street_density_km': 25911.383270470193,
 'clean_intersection_density_km': 172.0319992639525}
```

In [28]:

```
import pandas as pd
```

In [29]:

```
# unpack dicts into individiual keys:values
stats = ox.basic_stats(G, area=graph_area_m)
for k, count in stats["streets_per_node_counts"].items():
    stats["{}way_int_count".format(k)] = count
for k, proportion in stats["streets_per_node_proportions"].items():
    stats["{}way_int_prop".format(k)] = proportion

# delete the no longer needed dict elements
# del stats["streets_per_node_counts"]
# del stats["streets_per_node_proportions"]

# load as a pandas dataframe
pd.DataFrame(pd.Series(stats, name="value")).round(3)
```

Out[29]:

	value
n	88
m	182
k_avg	4.136364
edge_length_total	9269.973
edge_length_avg	50.933918
streets_per_node_avg	2.681818
streets_per_node_counts	{0: 0, 1: 10, 2: 12, 3: 62, 4: 4}
streets_per_node_proportions	{0: 0.0, 1: 0.11363636363636363, 2: 0.13636363...
intersection_count	78
street_length_total	5422.304
street_segment_count	110
street_length_avg	49.293673
circuity_avg	1.06439
self_loop_proportion	0.009091
node_density_km	420.522665
intersection_density_km	372.735998
edge_density_km	44298.110786
street_density_km	25911.38327
0way_int_count	0
1way_int_count	10
2way_int_count	12
3way_int_count	62
4way_int_count	4
0way_int_prop	0.0
1way_int_prop	0.113636
2way_int_prop	0.136364
3way_int_prop	0.704545
4way_int_prop	0.045455

In [30]:

```
import networkx as nx
```

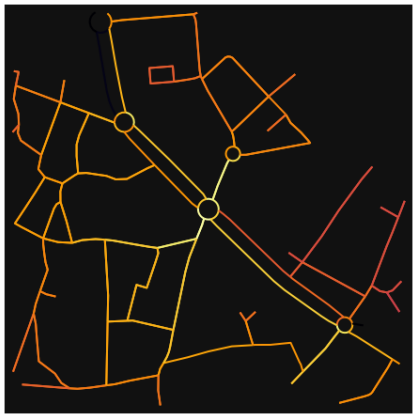
In [31]:

```
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
nx.set_edge_attributes(G, edge_centrality, "edge_centrality")
```

In [ ]:

In [32]:

```
# color edges in original graph with closeness centralities from line graph
ec = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=2, node_size=0)
```



In [33]:

```
# calculate betweenness with a digraph of G (ie, no parallel edges)
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
max_node, max_bc
```

Out[33]:

(4566811816, 0.43237102379043035)

In [34]:

```
# add the betweenness centraliy values as new node attributes, then plot
nx.set_node_attributes(G, bc, "bc")
nc = ox.plot.get_node_colors_by_attr(G, "bc", cmap="plasma")
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=30,
    node_zorder=2,
    edge_linewidth=0.2,
    edge_color="w",

)
```