ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

PhD THESIS

Zakaria BOUTARFA

A MULTIMODAL PUBLIC TRANSIT NETWORK DESIGN METHOD BASED ON HUB-AND-SPOKE INFRASTRUCTURE

DEPARTMENT OF COMPUTER ENGINEERING

ADANA, 2022

ABSTRACT

PhD. THESIS

A MULTIMODAL PUBLIC TRANSIT NETWORK DESIGN METHOD BASED ON HUB-AND-SPOKE INFRASTRUCTURE

Zakaria BOUTARFA

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER ENGINEERING

Supervisor : Prof. Dr. Mustafa GÖK
Year: 2022, Pages: 129
Jury : Prof. Dr. Mustafa GÖK
: Prof. Dr. Selma Ayşe ÖZEL
: Doç. Dr. Murat AKSOY
: Doç. Dr. Fatih KILIÇ
: Doç. Dr. Yasin KAYA

The quality of public transport in a modern city depends on the interaction of the different modes of transport. The efficiency of a multimodal public transport network (MPTN) can be improved by adapting the hub-and-spoke model. In this thesis, a two-phase method is presented to generate a hub-and-spoke network for MPTNs. In the first phase, the problem of clustering in MPTNs is addressed. A spatial clustering algorithm is presented that can process real urban MPTNs without reducing their size or dividing them into zones. The algorithm is tested on four large city datasets and compared with the popular spatial clustering algorithms. The largest test network (Sydney, 24063 nodes) was processed in less than a minute, and all clusters generated by the algorithm contain less than 100 nodes, while the clusters generated by the compared algorithms contain significantly more nodes. In the second phase, an algorithm based on multi-criteria decision making is presented, which simultaneously locates hubs and hub lines. The proposed method is tested with the MPTN of Greater London and the resulting hub-and-spoke network is compared with the Journey API provided by Transport for London. The results show that the total travel time is improved by 16.90% with a strict hubbing policy.

Key Words: Multimodal Public Transit Network Design, Spatial Clustering, Hub Location Problem, Hub Line Location Problem, Hub-and-spoke network.

ÖZ

DOKTORA TEZİ

HUB-AND-SPOKE ALTYAPISINA DAYALI ÇOK MODLU TOPLU ULAŞIM SİSTEM AĞI TASARIM YÖNTEMİ

Zakaria BOUTARFA

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Danışman : Prof. Dr. Mustafa GÖK
Yıl: 2022, Sayfa: 129
Jüri : Prof. Dr. Mustafa GÖK
: Prof. Dr. Selma Ayşe ÖZEL
: Doç. Dr. Murat AKSOY
: Doç. Dr. Fatih KILIÇ
: Doç. Dr. Yasin KAYA

Modern bir şehirde toplu taşımanın kalitesi, farklı ulaşım modlarının etkileşimine bağlıdır. Çok modlu bir toplu taşıma ağının (ÇMTTA) verimliliği, hub-and-spoke modeli uyarlanarak geliştirilebilir. Bu tezde, ÇMTTA'lara bir hub-and-spoke ağ oluşturmak için iki aşamalı bir yöntem sunulmaktadır. Birinci aşamada, ÇMTTA'larda kümelenme sorunu ele alınmaktadır. Gerçek kentsel ÇMTTA'ların boyutlarını küçültmeden veya bölgelere ayırmadan işleyebilen bir mekansal kümeleme algoritması sunulmaktadır. Algoritma, on büyük şehir veri seti üzerinde test edilmiş ve popüler mekansal kümeleme algoritmaları ile karşılaştırılmıştır. En büyük test ağı (Sidney, 24063 düğüm), bir dakikadan daha kısa bir sürede işlenmiş olup algoritma tarafından oluşturulan tüm kümeler 100'den az düğüm içerir. Karşılaştırılan algoritmalar tarafından oluşturulan kümeler önemli ölçüde daha fazla düğüm içermektedir. İkinci aşamada; çok kriterli karar vermeye dayalı, aynı anda hub ve hub hatlarını konumlandıran bir algoritma sunulmaktadır. Önerilen yöntem Greater London ÇMTTA'i ile test edilmiş olup ortaya çıkan hub ve bağlı ağ, Londra Ulaşım otoritesi (Transport for London) tarafından sağlanan Ulaşım uygulaması (Journey API) ile karşılaştırılmıştır. Sonuçlar, sıkı bir transfer politikası ile toplam seyahat süresinin %16.90 oranında iyileştirildiğini göstermektedir.

Anahtar Kelimeler : Çok modlu Toplu Taşıma Ağı Tasarımı, Mekansal Kümeleme, Hub Konum Problemi, Hub Hattı Konum Problemi, Hub ve Spoke ağ.

EXTENDED SUMMARY

Modern cities offer multiple public transportation modes such as bus services, tramways, subways, rapid bus, and inland waterway. The population growth and shift over the past years has increased the complexity of urban transportation planning (UTP). The economic, social, and environmental impacts of UTP are very high because of that researchers from different disciplines focus on this area. UTP involves the following main stages: 1) building the transportation infrastructure (roads, railways, tunnels, bridges, etc.); 2) route planning for different transportation modes; 3) setting up the frequencies of transportation services and scheduling time tables and drivers (Ceder & Wilson, 1986). These main stages are sequentially executed and interdependent. The first stage is performed by city planners and constrained by geographic topology. Once established it is very difficult and expensive to modify it. However, the second and third stages of planning are always open to optimization efforts and very significant returns can be obtained even by small improvements. For this reason, a vast amount of research effort has been dedicated to the second and third stages. Computer scientists handle these stages by developing graph models of transportation networks. These models are processed using efficient algorithms and high-performance computers.

Each type of transportation has its advantages and disadvantages. The public bus network is flexible and has a wide range, it can be established on any road. However, its capacity is limited and it is subject to traditional road problems such as speed limits, traffic, and traffic accidents. Rail networks, such as subways, on the other hand, are based on private infrastructure, which means they are not affected by traditional road problems and have high speed and capacity. But they are not flexible. Once a rail route is fixed, it is very expensive to change it. It is also subject to geographic constraints; a line cannot be laid out everywhere, so its reach is limited. The operator's goal is to take advantage of the different modes by combining them

into a well-coordinated multimodal transportation network that provides better service to passengers.

Travel on a multimodal network involves transferring in the middle. While transferring, the walking distance and waiting time can be very long. For this trip to be convenient for the passenger, the total travel time, including transfers, should be shorter than for a single mode trip. In order to plan this network efficiently, decision makers need to consider many criteria, such as: the interaction of the different modes of transport, the provision of sufficient capacity to meet demand, the synchronization of the networks' schedules, the accessibility of the network for the different population groups, including passengers with special needs and the elderly, the availability of parking, etc.

A short travel time is not the only criterion by which passengers make their choices. In general, excessively long transfer times are unfavorable, especially when weather conditions are unfavorable. The cost of the trip is also an important factor (parking ticket, transportation fees). Network operators usually provide a digital platform with several routes to choose from to help passengers decide on a trip.

The existing literature on Urbain Multimodal Transportation Networks UMTN design problem can be divided into three categories: Modal Split Optimization, Multicriteria Shortest Path, and Multimodal Public Transportation Networks (MPTN). The first two categories include both public and private transportation networks, while the third category deals only with public transportation networks.

Modal split optimization is concerned with solving the problem posed by competition between the public transportation network and private transportation sharing the same routes. Especially in busy corridors, where bus network operators tend to increase their frequency to generate more revenue. The relevant literature focuses on redesigning routes, frequencies, and schedules of the bus network to achieve good synchronization with private networks and improve the overall performance of the network.

The multicriteria shortest path problem is to find the best path based on passengers' preferences. In this category, it is assumed that the UMTN has already been created. The algorithms created here are used in the platforms provided by the operators to help passengers choose the route.

The third category focuses only on public transportation networks. It targets large cities where public transportation is used by the majority of the population, including private car owners, such as New York, London, Istanbul, etc. Since rail-based public transportation networks are fixed, similar to the first group, only bus routes are redesigned, but the frequencies and schedules of all modes are synchronized. A new trend in this area is the use of hub-and-spoke infrastructure to model the network.

In this thesis, the focus on the third category, in particular on the use of hub-and-spoke networks as infrastructure for MPTN. Applying this topology to the MPTN design, the hubs are located in rail-based transportation networks and the bus lines are used as spoke lines and allocated to the hubs. This architecture simplifies the problem to some extent and can support the development of feasible solutions. Moreover, literature reports that a trip on a well-developed hub-and-spoke network can compete with private transport (Daganzo, 2010). In this way, more private car users are attracted to public transit.

Most of the research in this area uses clustering techniques to locate the hubs. The quality of solutions at all levels of MPTN design depends on the effectiveness of the clustering algorithms. However, several problems associated with the clustering techniques used in the existing literature are observed.

First, to locate hubs, most works consider predefined clusters, usually based on administrative or geographical aspects. Another problem is the required number of clusters as input. In addition, to cope with the high computation time, simplifications are made in the test datasets. Most clustering solutions are problem-specific and require an initial step to define the parameters for the test network.

Therefore, it is often not possible to estimate the performance of the algorithm on other networks.

In addition, the datasets used to test the performance of these methods are very small. In comparison, the MPTN datasets are relatively large compared to these benchmarks. For example, the city of Sydney in Australia has an MPTN with 24063 nodes, where each node is a stop of a transportation mode (Kujala et al., 2018).

Another problem is that most studies focus only on the hub location problem and assume a fully connected hub network. This scenario is unrealistic because it is almost impossible for rail systems to have so many connections.

Locating hubs on rail-based networks creates an equality problem. Typically, only developed areas with high demand can benefit from a hub network, leaving out large portions of the city, creating an accessibility problem for these areas.

In this study, a two-phase solution to these problems is proposed. In the first phase, a clustering algorithm is presented. The algorithm is called Spatial Clustering for Public Transit (scpt), it does not require any number of clusters or predefined cluster zones as input. It can handle any network without simplifying the dataset. In addition to these advantages, the clusters generated by the algorithm are uniformly distributed across the networks and have feasible sizes. The algorithm uses the Delaunay triangulation method to detect the proximity between data points. Then, the edges are gradually removed based on statistical thresholds until all clusters are identified.

In the second phase, a heuristic algorithm for finding hubs and hub lines is presented. It focuses on maximizing the network coverage. The growing network model is used to find the root hub, then the other hubs are located based on the optimality of the existing line connections. Finally, a bus network based on a bus network is built in the areas affected by inequality.

The performance of the clustering algorithm is tested on MPTN datasets of Athens, Paris, Sydney, Brisbane, Berlin, Bordeaux, Toulouse, Helsinki, Lisbon and

Winnipeg cities. The whole methodology is tested on the Greater London MPTN and compared with the Journey API provided by Transportation for London (TFL).

The results show that the proposed method not only significantly improves the overall travel time, but also provides high network accessibility in terms of coverage. Moreover, the method is extremely flexible: the clustering algorithm showed similar results in all cities, regardless of the different topologies. In addition, it provides a parameter called alpha that can be used to control the size of the clusters. The techniques for evaluating hubs and hub lines can be easily substituted by decision makers depending on their preferences.

GENİŞLETİLMİŞ ÖZET

Modern şehirler, otobüs hizmetleri, tramvaylar, metrolar, hızlı otobüs ve iç su yolu gibi birden fazla toplu taşıma modu sunar. Son yıllardaki nüfus artışı ve kayması, kentsel ulaşım planlamasının (KUP) karmaşıklığını artırmıştır. KUP'nin ekonomik, sosyal ve çevresel etkileri, farklı disiplinlerden araştırmacıların bu alana odaklanması nedeniyle çok yüksektir. KUP aşağıdaki ana aşamaları içerir: 1) ulaşım altyapısının (karayolları, demiryolları, tüneller, köprüler, vb.) inşa edilmesi; 2) farklı ulaşım modları için rota planlaması; 3) ulaşım hizmetlerinin sıklıklarının ayarlanması ve zaman çizelgelerinin ve sürücülerin programlanması (Ceder & Wilson, 1986). Bu ana aşamalar sırayla yürütülür ve birbirine bağlıdır. İlk aşama, şehir plancıları tarafından gerçekleştirilir ve coğrafi topoloji ile sınırlandırılır. Bir kez kurulduktan sonra onu değiştirmek çok zor ve pahalıdır. Ancak planlamanın ikinci ve üçüncü aşamaları her zaman optimizasyon çalışmalarına açıktır ve küçük iyileştirmelerle bile çok önemli getiriler elde edilebilir. Bu nedenle, ikinci ve üçüncü aşamalara büyük miktarda araştırma çabası ayrılmıştır. Bilgisayar bilimciler bu aşamaları ulaşım ağlarının çizge modellerini geliştirerek ele alırlar. Bu modeller, verimli algoritmalar ve yüksek performanslı bilgisayarlar kullanılarak işlenir.

Her ulaşım türünün avantajları ve dezavantajları vardır. Halk otobüsü ağı esnektir ve geniş bir yelpazeye sahiptir, her yola kurulabilir. Ancak kapasitesi sınırlıdır ve hız sınırları, trafik, trafik kazaları gibi geleneksel yol sorunlarına maruz kalmaktadır. Metro gibi raylı ağlar ise özel altyapıya dayalıdır, yani geleneksel yol sorunlarından etkilenmezler ve yüksek hız ve kapasiteye sahiptirler. Ama esnek değiller. Bir demiryolu güzergahı bir kez sabitlendiğinde, onu değiştirmek çok pahalıdır. Ayrıca coğrafi kısıtlamalara tabidir; her yerde bir demiryolu güzergahı düzenlenemez, bu nedenle erişimi sınırlıdır. Operatörün amacı, farklı modları yolculara daha iyi hizmet sağlayan iyi koordine edilmiş çok modlu bir ulaşım ağında birleştirerek yararlanmaktır.

Çok modlu bir ağda seyahat, ortada aktarmayı içerir. Transfer sırasında yürüme mesafesi ve bekleme süresi çok uzun olabilir. Bu yolculuğun yolcu için uygun olması için, transferler dahil toplam seyahat süresinin tek modlu bir seyahatten daha kısa olması gerekir. Bu ağı verimli bir şekilde planlamak için, karar vericilerin aşağıdakiler gibi birçok kriteri dikkate alması gerekir: farklı ulaşım modlarının etkileşimi; talebi karşılamak için yeterli kapasitenin sağlanması; ağların programlarının senkronizasyonu; özel ihtiyaçları olan yolcular ve yaşlılar dahil olmak üzere farklı nüfus grupları için ağın erişilebilirliği; otopark mevcudiyeti vb. Yolcuların seçimlerini yaparken kullandıkları tek kriter kısa seyahat süresi değildir. Genel olarak, özellikle hava koşulları elverişsiz olduğunda, aşırı uzun transfer süreleri elverişsizdir. Seyahatin maliyeti de önemli bir faktördür (otopark bileti, ulaşım ücretleri). operatörler genellikle yolcuların bir seyahate karar vermelerine yardımcı olmak için aralarından seçim yapabilecekleri çeşitli rotalara sahip dijital bir platform sağlar.

KUP tasarım problemi üzerine mevcut literatür üç kategoriye ayrılabilir: Modal Bölünmüş Optimizasyon, Çok Kriterli En Kısa Yol ve Çok Modlu Toplu Taşıma Ağları (ÇMTTA). İlk iki kategori hem toplu hem de özel ulaşım ağlarını içerirken, üçüncü kategori sadece toplu taşıma ağlarını içermektedir.

Modal bölünmüş optimizasyon, toplu taşıma ağı ile aynı rotaları paylaşan özel ulaşım arasındaki rekabetin yarattığı sorunu çözmekle ilgilenir. Özellikle otobüs ağı operatörlerinin daha fazla gelir elde etmek için frekanslarını artırma eğiliminde olduğu yoğun koridorlarda. İlgili literatür, özel ağlarla iyi bir senkronizasyon sağlamak ve ağın genel performansını iyileştirmek için otobüs ağının rotalarını, frekanslarını ve programlarını yeniden tasarlamaya odaklanmaktadır.

Çok kriterli en kısa yol problemi, yolcuların tercihlerine göre en iyi yolu bulmaktır. Bu kategoride, ÇMTTA'nın zaten oluşturulmuş olduğu varsayılır. Burada oluşturulan algoritmalar, yolcuların rotayı seçmelerine yardımcı olmak için operatörler tarafından sağlanan platformlarda kullanılmaktadır.

Üçüncü kategori sadece toplu taşıma ağlarına odaklanmaktadır. New York, Londra, İstanbul gibi özel araç sahipleri de dahil olmak üzere nüfusun çoğunluğu tarafından toplu taşımanın kullanıldığı büyük şehirleri hedefler. Demiryoluna dayalı toplu taşıma ağları birinci gruba benzer şekilde sabit olduğundan, sadece otobüs rotalar yeniden tasarlandı, ancak tüm modların frekansları ve programları senkronize edilmelidir. Bu alandaki yeni bir trend, ağı modellemek için hub-and-spoke altyapısının kullanılmasıdır.

Bu tezde, üçüncü kategoriye, özellikle de ÇMTTA için altyapı olarak hub-and-spoke ağların kullanımına odaklanıyoruz. Bu topolojiyi ÇMTTA tasarımına uygulayarak, hub'lar demiryolu tabanlı ulaşım ağlarında bulunur ve otobüs hatları, spoke hatlar olarak kullanılır ve hub'lara tahsis edilir. Bu mimari, sorunu bir dereceye kadar basitleştirir ve uygun çözümlerin geliştirilmesini destekleyebilir. Ayrıca, literatür, iyi gelişmiş bir hub-and-spoke üzerinde bir seyahatin özel ulaşım ile rekabet edebileceğini bildirmektedir (Daganzo, 2010). Bu sayede daha fazla özel araç kullanıcısı toplu taşımaya ilgi duymaktadır.

Bu alandaki araştırmaların çoğu, Hub'lar yerini belirlemek için kümeleme tekniklerini kullanır. ÇMTTA tasarımının tüm seviyelerindeki çözümlerin kalitesi, kümeleme algoritmalarının etkinliğine bağlıdır. Ancak, mevcut literatürde kullanılan kümeleme teknikleriyle ilgili birkaç problem belirledik.

İlk olarak, Hub'ların yerini belirlemek için çoğu çalışma, genellikle idari veya coğrafi yönlere dayalı olarak önceden tanımlanmış kümeleri dikkate alır. Diğer bir problem, girdi olarak gerekli sayıda kümedir. Ayrıca, yüksek hesaplama süresi ile başa çıkmak için test veri setlerinde basitleştirmeler yapılmıştır. Çoğu kümeleme çözümü soruna özeldir ve test ağının parametrelerini tanımlamak için bir ilk adım gerektirir. Bu nedenle, diğer ağlarda algoritmanın performansını tahmin etmek çoğu zaman mümkün değildir.

Ayrıca bu yöntemlerin performansını test etmek için kullanılan veri setleri çok küçüktür. Karşılaştırıldığında, ÇMTTA veri setleri, bu karşılaştırma ölçütlerine kıyasla nispeten büyüktür. Örneğin, Avustralya'daki Sidney şehri, her düğümün bir

ulaşım modunun durağı olduğu 24063 düğümlü bir ÇMTTA'ya sahiptir (Kujala ve diğerleri, 2018).

Diğer bir sorun ise, çoğu çalışmanın yalnızca hub konumu sorununa odaklanması ve tam bağlantılı bir hub ağını varsaymasıdır. Bu senaryo gerçekçi değil çünkü raylı sistemlerin bu kadar çok bağlantıya sahip olması neredeyse imkansız.

Demiryolu tabanlı ağlarda merkezlerin bulunması bir eşitlik sorunu yaratır. Tipik olarak, yalnızca yüksek talebe sahip gelişmiş bölgeler bir hub ağından faydalanabilir, bu da şehrin büyük bölümlerini dışarıda bırakarak bu alanlar için erişilebilirlik sorunu yaratır.

Bu çalışmada, bu sorunlara iki aşamalı bir çözüm öneriyoruz. İlk aşamada, bir kümeleme algoritması sunulmaktadır. Algoritma Toplu Taşıma için Uzamsal Kümeleme (scpt) olarak adlandırılır, girdi olarak herhangi bir sayıda küme veya önceden tanımlanmış küme bölgesi gerektirmez. Veri kümesini basitleştirmeden herhangi bir ağı işleyebilir. Bu avantajlara ek olarak, algoritma tarafından oluşturulan kümeler ağlar arasında eşit olarak dağıtılır ve uygun boyutlara sahiptir. Algoritma, veri noktaları arasındaki yakınlığı tespit etmek için Delaunay üçgenleme yöntemini kullanır. Ardından, tüm kümeler tanımlanana kadar kenarlar, istatistiksel eşiklere dayalı olarak kademeli olarak kaldırılır.

İkinci aşamada, hub ve hub hatlarını bulmak için sezgisel bir algoritma sunulmaktadır. Ağ kapsama alanını en üst düzeye çıkarmaya odaklanır. Büyüyen ağ modeli, kök hub'ı bulmak için kullanılır, ardından diğer hub'lar, mevcut hat bağlantılarının optimalliğine dayalı olarak yerleştirilir. Son olarak, eşitsizlikten etkilenen alanlarda bir otobüs ağına dayalı bir otobüs ağı kurulur.

Kümeleme algoritmasının performansı Atina, Paris, Sidney, Brisbane, Berlin, Bordeaux, Toulouse, Helsinki, Lizbon ve Winnipeg şehirlerindeki ÇMTTA veri setleri üzerinde test edilmiştir. Metodolojinin tamamı Greater London ÇMTTA üzerinde test edilmiş ve Transport for London (TFL) tarafından sağlanan Journey API ile karşılaştırılmıştır.

Sonuçlar, önerilen yöntemin yalnızca genel seyahat süresini önemli ölçüde iyileştirmekle kalmayıp, aynı zamanda kapsama açısından yüksek ağ erişilebilirliği sağladığını göstermektedir. Ayrıca, yöntem son derece esnektir: kümeleme algoritması, farklı topolojilerden bağımsız olarak tüm şehirlerde benzer sonuçlar göstermiştir. Ayrıca, kümelerin boyutunu kontrol etmek için kullanılabilecek alfa adlı bir parametre sağlar. Hub'ları ve hub hatlarını değerlendirme teknikleri, tercihlerine bağlı olarak karar vericiler tarafından kolayca değiştirilebilir.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor and chair of my committee, Prof. Dr. Mustafa Gök for his support, invaluable patience and advise that helped me through my research journey. I also could not have undertaken this journey without my defense committee, who generously provided knowledge and expertise. Additionally, this endeavor would not be possible without the generous support of YTB.

Words cannot express my gratitude to my dear wife Alexandra, who provided continues love and moral support throughout the ups and downs of my work. Many thanks to son Adam, for bringing out of me an Energy I didn't know I had. Also, I'm thankful to Adan for making sleep a luxury long forgotten.

I am also grateful to İzzet kılınç for his friendship and helping me during my stay in Adana. Lastly, I would be remiss in not mentioning my family, especially my parents, their belief in me has kept my spirits and motivation high during this process.

CONTENTS **PAGE**

LIST OF TABLES **PAGES**

LIST OF FIGURES **PAGE**

1 INTRODUCTION

1.1 Urban Multimodal Transportation Networks (UMTN)

Transportation is the backbone of mobility in urban areas. People's daily activities, such as work, education, health, shopping, sports, and entertainment, etc., depend heavily on transportation. Normally, people tend to find a job near their home or a home near their workplace to avoid being stuck in traffic. Unfortunately, this is not always the case. Rapid population growth and the increasing use of private vehicles have put more strain on the transportation network, such as the increase in traffic and traffic accidents, noise pollution, carbon emission, and the difficulty of finding a parking space.

City planners try to solve these problems by building rail-based transportation systems. These modes of transportation operate on private infrastructure, have high capacity, can travel at high speeds, and are not subject to traditional traffic problems. However, they are expensive and cannot be built everywhere due to geographic limitations. In contrast, the public bus network (PBN) and personal transportation (PT) such as car, bicycle, motorcycle are more flexible and can reach any part of the city, but have low capacity and are subject to traditional road problems. Urban planners try to exploit the strengths of the different modes by combining them into a multimodal transportation network.

Traditionally, each mode of transportation is planned individually. The planning process includes the following main phases:

- Construction of transportation infrastructure (roads, railways, tunnels, bridges, etc.).
- Route planning for the different modes.
- Determination of service frequencies and planning of schedules and drivers (Ceder & Wilson, 1986), this step is for public transportation only.

These main phases are carried out sequentially and are interdependent. The first phase is performed by urban planners and is constrained by geographic topology. Once established, it is very difficult and expensive to change. However, the second and third phases of planning are always open to optimization efforts, and significant gains can be made by even minor improvements.

However, in the case of the UMTN, all modes of transportation must be considered in the planning phase. For this network to be efficient, planners must ensure optimal interaction between modes and synchronization of their schedules.

1.2 UMTN Design Problem

A multimodal trip from origin A to destination B, where the passenger uses the bus and the subway, involves a transfer in the middle at station C. During the transfer, the walking distance and the waiting time can be very long. In order to make this trip convenient for the passenger, the total travel time, including the transfer, should be shorter than that of a trip using a single mode of transportation. To plan this network efficiently, decision makers must answer many questions, such as:

- Where do busses and subways interact?
- Is there enough capacity to meet demand?
- When should vehicles arrive and depart?
- Is there a walking path during the transfer?
- Is there suitable parking for PT? etc.

A short travel time is not the only criterion by which passengers make their choices. Generally, transfers that are too long are unfavorable, especially when weather conditions are not suitable. The cost of travel is also an important factor (parking ticket, transportation fees). Network operators usually provide a digital platform to help passengers decide which trip to take.

Figure 1.1 shows the multimodal transportation network of the city of Istanbul, Turkey. From the figure, it is clear how complex the network is and how much choice is available when a passenger wants to travel from A to B. Therefore, designing an efficient network and providing appropriate tools for users to benefit from it not only facilitates the daily flow for travelers, but can also bring economic benefits for operators.

Figure 1.1. UMTN of Istanbul, Turkey, map from “ÖPNVKarte”, licences:"Map memomaps.de CC-BY-SA, map data Openstreetmap ODbL".

2 LITERATURE REVIEW

Researchers have addressed many aspects of the UMTN design problem, which can be grouped into three main categories: problem-specific optimization, multi-criteria routing, and multimodal public transportation networks (MPTN). The first two categories cover both public and private transportation networks, while the third category deals only with public transportation networks.

Most of the optimization work in the literature focuses on redesigning the bus network routes, frequencies, and schedules with a certain objective (s) so that it is well synchronized with the private networks and improves the overall performance of the network.

The second category focuses on the multi-criteria shortest path problem. Here, it is assumed that the UMTN is already established, and the aim is to find the best path based on the users' preferences. These are the algorithms used in the platform provided by the operators to help passengers choose the route.

The third category focuses exclusively on the synchronization of public transport networks. It targets large cities where public transportation is used by the majority of the population, including private car owners, such as New York, London, Istanbul, and so on. Since the rail-based public transportation networks are fixed, similar to the first group, only the bus routes are redesigned, but the frequencies and schedules of all the modes are synchronized. A new trend in this area is the use of hub-and-spoke infrastructure to model the network.

The corresponding papers in each category are listed in Tables 2.1-2.3. In all tables, the first column contains the reference to the research, the third column contains the solution to the problem proposed by the authors, and the last column contains the network used to test the performance of the proposed solution. In the first and second tables, the third column represents the modes involved in the methodology. The second column in the first and third table represents the objectives of the study, while in the second table it indicates the criteria for the routing problem.

The fourth column in the third table represents the clustering method used in the hub location.

Table 2.1. Modal split methodologies for UMTN

Study	Objective	Solution	Modes	Dataset
(Cipriani et al., 2006)	Min. total cost.	Heuristic	Bus, Car	49 nodes
(Beltran et al., 2009)	Min. total cost.	Heuristic, Genetic Algorithm	Bus, Green bus, Car	49 nodes
(Cipriani et al., 2012)	Min. total cost.	Heuristic, Parallel Genetic Algorithm	Bus, Rapid Rail, Tramway, Car	1300 nodes
(Fan & Machemehl, 2006a)	Min. operators' cost, users' cost and non-covered trip cost	Genetic Algorithm	Bus, Car	160 nodes
(Fan & Machemehl, 2006b)	Min. operators' cost, users' cost and non-covered trip cost	Simulated Annealing	Bus, Car	160 nodes
(Fan & Machemehl, 2008)	Min. operators' cost, users' cost and non-covered trip cost	Tabu Search	Bus, Car	160 nodes
(Wan & Lo, 2009)	Max. social benefit and min cost.	Heuristic	Bus, Car, Subway	12 nodes
(M. Gallo et al., 2011)	Optimal frequency	Heuristic	Bus, Car, Walk	NA
(Song et al., 2017)	Max social benefit	Heuristic	Car, subway	28 nodes
(Z. Liu et al., 2018)	Min. total travel time	Exact methodology	Car, train	28 nodes

NA: not available.

Table 2.2. Recent multicriteria shortest path studies for UMTN

Study	Criteria	Solution	Modes	Dataset
(Lozano & Storchi, 2001)	Min. travel time & transfers	Heuristic	PN, SW, BN, Bike, Motorbike, Car.	21 nodes
(Bielli et al., 2006)	Min. travel time.	Heuristic	Car, SW, BN, PN	1000 nodes
(Abbaspour & Samadzadegan, 2010)	Best path	Heuristic GA	BN, SW, PN	4808 nodes
(L. Liu et al., 2014)	Min. travel time & transfers	Heuristic	NA	40000 nodes
(Dib et al., 2017)	Min. trip time, cost, transfer and walking	GA, VNS	SW, TW, BN, PN	17950 nodes
(Dib et al., 2018)	Min. trip time, cost, transfer and walking	GA, HL	SW, TW, BN, PN	17950 nodes
(Dalkılıç et al., 2017)	Min. trip time, transfer	Heuristic	SW, BN, PN	NA
(López & Lozano, 2020)	Mode choice, nbr transfers, walking time	Heuristic	PN, SW, BN, Bike, Motorbike, Car.	39871 nodes
(Peng et al., 2021)	Trip cost, nbr transfers	Metaheuristic	BN, SW, PN	30 nodes

NA: not available.

Table 2.3. Recent studies that use hub and spoke model for MPTN design problems

Study	Objective	Solution	Clustering	Dataset
(Sung & Jin, 2001)	min. network cost	Integer programming.	Predetermined number of clusters	50 nodes, Simulated
(J. Yu et al., 2008)	Min total traveling Time	Mixed nonlinear integer program	Predetermined clusters	Aggregated to 58 zones (centroid)
(J. Yu et al., 2009)	Min total traveling Time	Mixed integer program	Predetermined clusters	Aggregated to 58 zones (centroid)
(Z. Wang & Chen, 2012)	Max profit, passenger density and coverage.	Heuristic	Predetermined clusters	Aggregated to 12 zones (centroid)
(B. Yu et al., 2013)	Max demand served/cost. Min overlapped demand	Heuristic	Clustering based on a predefined radius	3000 nodes aggregated to 998 nodes
(Yuan & Yu, 2018)	Total system performance	Genetic Algorithm and Augmented Lagrangian Dual alg.	Predetermined clusters	Aggregated to 58 zones (centroid)
(Huang et al., 2018)	Increase network efficiency. Min total cost.	Clustering by Fast search. Heuristic for TSP. BCO for frequency.	Finds clusters automatically.	NA

NA: not available.

2.1 Modal Split Methodologies for UMTN Design Problem

The modal split consists of the ratio of users traveling by a particular mode of transportation. A user's decision to use a particular mode of transportation depends on several criteria. Often, users who own a motorized vehicle prefer it for their trips, and non-owners of a motorized private vehicle are more likely to use public transit. However, other factors such as fuel costs, congestion, and weather conditions can also influence a user's choice if there is a better alternative. For more details on user choice, the readers are referred to (Santos et al., 2013). Urban planners agree that the increasing use of private vehicles has negative impacts on both the environment and infrastructure. Therefore, alternative solutions must be found to attract private vehicle users to public transportation. The situation is exacerbated because public busses use the same infrastructure as private vehicles, especially on busy corridors where bus frequency is usually high. Therefore, it is an important task for urban planners to design public bus routes to influence user choice. This issue has attracted the attention of many researchers. In the following, the relevant works in this field are presented.

Cipriani et al. proposed a heuristic solution to the UMPT design problem with dynamic demand that incorporates public bus and private car modes (Cipriani et al., 2006). Their methodology solves the problem in three phases. In the first phase, a procedure is presented that generates a set of feasible bus routes. This procedure generates three different types of routes. The first type is for passenger's interests and includes routes based on shortest path. The second type is for operator's interest and includes routes based on maximizing capacity usage and optimizing cost. In contrast to the first type, it generates routes that include transfer connections, this category generates two subtypes of routes: main routes and feeder routes. The third type, is the lines of the existing network. In the second phase, a genetic algorithm (GA) is used to determine the optimal bus routes and their relative frequencies. The routes generated in the first phase are used as the initial population of GA. Each route is considered as one chromosome. Then, tournament selection

and one-gene mutation are applied to each chromosome. Finally, the individuals are mutated to generate new progeny and the process is repeated with the new generation as input until the termination criteria are met. In the last phase, a heuristic is used to enhance the performance of the network. The algorithm takes the lines generated by GA as input. It traverses all the routes and improves the routes one by one by checking if the route can be extended, shortened. The cost of the changes are computed for the entire network. This method has been tested on the real network of the city of Rome, Italy, with 49 nodes (Cipriani et al., 2006).

In a related work (Beltran et al., 2009), the previous method was extended by including a set of eco-friendly vehicles. Their method has the same two-stage structure as the previous work. The main difference is the optimization of two types of bus networks. One is the traditional bus network and the other is the eco-friendly bus network constrained by a limited fleet.

In a recent work (Cipriani et al., 2012), the authors tackled a broader version of the network with 1300 nodes. Unlike the previous work, a two-stage solution was proposed with the goal of minimizing the total operating cost. In the first stage, the same type of feasible routes are generated as in the previous work (Cipriani et al., 2006). However, in this version, the demand is static and the passenger flow is considered when inserting connections. In the second phase, due to the large size of the routes, a parallel GA was implemented to generate optimal bus routes and their frequencies.

In (Fan & Machemehl, 2006a), a mixed-integer multi-objective formulation for solving the PBN design problem was presented. The objectives of this formulation are to minimize the operating cost, travel time, and cost of non-covered trips. A three-step method was developed to solve this problem. In the first step, the K shortest path algorithms of Dijkstra and Yen are used to generate a set of feasible routes constrained by the length of the route. For Yen's algorithm, the algorithm was modified to find all accepted paths of the specified length instead of k paths. In the second step, the set of routes generated in the first step is filtered to determine the

demand between stations and the route frequencies. All routes and parameters generated in the first two steps are used as input to GA, which searches for a suboptimal set of lines. A simulated network with 160 nodes is used to test the proposed method.

In a parallel work, (Fan & Machemehl, 2006b) presented a different method for solving the PBN design problem. Instead of GA, Simulated Annealing (SA) is used in the third step of the solution process. In the experiments, a comparison between the solutions generated by SA and GA showed that the results of SA are more optimal.

In a later work following the same principles as the previous related work (Fan & Machemehl, 2008) presented a method based on Tabu Search (TA). In the experiments, a comparison between the solutions generated by TA and GA shows that the results of TA are more optimal.

(Wan & Lo, 2009) presented a bi-level mathematical formulation for UMPT that takes into account traffic and passenger choice. At the upper level, a heuristic algorithm is presented to determine the demand matrix of PBN. This algorithm removes connections that do not provide any benefit until termination, and the demand matrix is based on the remaining nodes. At the lower level, optimal bus routes and their frequencies are generated. A mixed-integer linear program is introduced and solved using the commercial software CIPLEX. To test the proposed method, a simulated benchmark with 12 nodes and three modes is used: Private car, metro and bus.

In the work of (M. Gallo et al., 2011), a heuristic was presented that consists of five steps. The objective is to optimize the frequencies of bus networks in the presence of dynamic demand. It is assumed that the bus network is given, and the work focuses only on generating the frequencies of bus routes considering the other modes. In the first step of the algorithm, only the nearest solutions are considered and no constraints are applied. In the second step, all frequencies are set to the best frequencies found in the previous step. In the third step, the neighborhood search

algorithm (NS) is used to find the frequencies of the bus lines, similar to the previous step, with no constraints applied. In the fourth step, the values that minimize the objective function are selected from the results. In the last step, the optimal solutions are found with NS considering all constraints. In order to simulate a realistic scenario, private and walking networks are also considered in the optimization model. The proposed methodology was tested with the real network of Salerno, Italy, with 40 bus routes.

(Song et al., 2017) presented a methodology for planning park-and-ride locations, including determining the optimal frequencies for the transportation network. Park-and-ride is the location of a series of parking lots with access to public transportation such as buses or subways. The goal of this method is to encourage private vehicle users to transfer to a public transit line during their trip. Normally, parking fees are a part of the travel cost for the passenger. The authors presented a mathematical formulation with equilibrium constraints and solved the problem using a two-phase heuristic algorithm. The performance of their method was tested on a benchmark network with 28 nodes.

(Z. Liu et al., 2018), proposed a new method for the problem of designing park-and-ride networks. Finding a parking lot near a bus or train station is challenging in itself and can be very costly, especially in city centers. For this reason, the authors propose placing parking lots in suburban areas and connecting them to desired train stations via rapid bus routes. An exact method to solve the proposed formulation was implemented and tests were conducted in the network of Sioux Falls, South Dakota, USA, with 28 nodes.

2.2 Multicriteria Shortest Path Problem in UMTN

The multicriteria shortest path problem (MCSPP) in UMTN is to find the shortest path based on user preferences. The shortest path does not necessarily have to be the fastest or the cheapest path to reach the destination. Especially in today's urban transportation networks, which are composed of different modes of

transportation, finding the best way to a destination is very complex. Urban planners are trying to solve this problem by providing a digital platform that offers multiple paths to a destination. An example of a multimodal network is shown in Figure 2.1. Let's consider a journey that starts from O to D in this figure. An algorithm that find the best path should take account of multiple scenarios for different passengers. For example, the question whether the user intend to use a private car, public transportation, or both should be addressed. Also, transferring, walking, parking, and the cost of travel are commonly used search criteria.

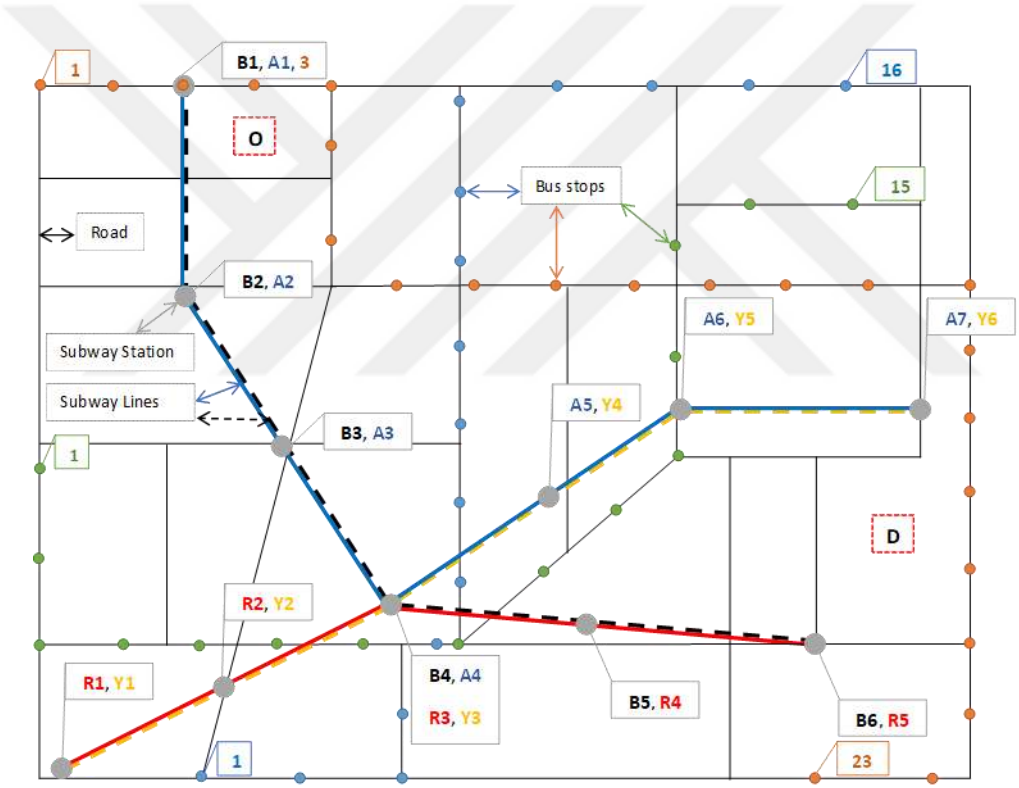


Figure 2.1. Example of a multicriteria trip in a multimodal network

In Table 2.4, an example of a trip from origin *O* to destination *D* in Figure 2.1. is provided. In this scenario, it is assume that the user does not own a car and must use public transportation. Also, it is assumed that the user leaves at 8:00 am

Table 2.4. Example of multimodal trips based on Figure 2.1, from O to D.

Trip	Trip-legs	Time (min)	Cost (₺)
Path 1	O(walk)→B1-B6(subway)→(walk)D	5→15→12 = 32	12
Path 2	O(walk)→A1-A7(subway)→(walk)D	5→17→8 = 30	12
Path 3	O(walk)→3-18(bus)→(walk)D	5→20→5 = 40	7
Path 4	O(walk)→A1-A4(subway)→Y3-Y6(subway)→(walk)D	5→10→$5^{transfer}$→10→5 = 35	24

and must reach the destination at 8:35 am. It is clear that trips 1 and 2 can reach *D* in a convenient time, while trip 3 can be reached in a strict time, but any extra waiting in transferring can cause delays. If the user has a budget of less than 12₺, the only suitable trip is 3, even if it is late. In practice, lines using the same rail infrastructure depart at different times. If the schedules of the individual lines are included, the suitability of the trips shifts accordingly and this simple example becomes more complex. Building such a tool to accept requests from millions of users at any given time, taking into account different scenarios and preferences, requires robust data modeling.

Typically, each mode of transportation is modeled as a graph. UMTN is the collection of these graphs. Let G be the graph representing the network such that G= (V, E), the set of vertices V= $\{v_1, v_2,...v_N\}$ represents stops, stations, or intersections, and E= $\{e_1, e_2, ... e_M\}$ stands for the set of edges. An edge is the line that connects two stops, stations, side paths, or intersections. The following transportation networks are included in this example: Road network (RN), Bus network (BN), Rapid bus (RB), Subway (SW), Tramway (TW), Inland Waterway (IW) and Pedestrian network (PN).

RN represents the main transport infrastructure of an urban area, where each vertex represents an intersection and each edge represents a road. BN represents the public bus network which is a layer on RN.

In the literature, many researchers have used this model to solve the MCSPP problem. A summary of these works can be found in Table 2.2. This table is similar

to Table 2.1. except that the fourth column of the table, shows the abbreviations of the modes mentioned in the above paragraph.

(Lozano & Storchi, 2001) proposed a label-based algorithm to solve the shortest path problem in UMTN. The algorithm is a modified version of the shortest path studied in (G. Gallo & Pallottino, 1988). The algorithm generates a set of feasible paths that satisfy several criteria, such as the travel cost and the number of transfers involved. The users select amongst the generated solutions. The proposed methodology was tested with a benchmark network with 21 nodes.

In (Bielli et al., 2006), a tool was implemented that helps the user to perform a multicriteria journey. The main contribution of the work is an algorithm based on the k-shortest paths algorithm. The algorithm generates paths constrained by the user's preferences, such as the total travel time, the maximum number of transfers, and the modes of transportation. The algorithm can be used for both city networks and intercity networks. Test were performed on networks that consist of up to 1000 nodes.

In (Abbaspour & Samadzadegan, 2010), a GA based algorithm for solving the multimodal shortest path in public transportation was presented. The multimodal network is modeled as a set of subgraphs, where each subgraph represents a public transportation network and these networks are connected by walking edges. The algorithm solves the problem in five phases. First, the paths are represented by the chromosomes of the GA. Since the number of nodes in a path is not known apriori, the size of the chromosome is adaptive. The genes of the chromosome are randomly generated. In the second stage, half of the population with the best performance is kept, while the rest are removed from the mating pool. The third step consists of selecting two chromosomes using random selection. The fourth step consists of generating the offspring using crossover methods with one or two points. Replacing a gene can generate cycles and invalidate pathways. Therefore, in the fifth phase, the authors used a method that scans chromosomes to prevent this issue and reverse the crossover. This algorithm was tested in a dataset of the city of Tehran, Iran, with 45

metro stations and 4763 bus stops. 500 nodes were used to generate the paths, half as the starting point of the trip and the other half as the destination.

In (L. Liu et al., 2014), an exact labeling algorithm was presented to solve the shortest path problem in UMTN. In traditional labeling algorithms for monomodal shortest paths, a label is set for each node. The authors adapted this algorithm to multimodal paths, where multiple labels are assigned to a node based on the paths that pass through the node. A second algorithm was presented that uses reverse labeling to define the final path. The main objective of this algorithm is to update the path starting from the destination according to the arrival delays. The methodology were tested with a variety of simulated grid-like networks with up to 40000 nodes.

Dib et al presented a metaheuristic composed of GA and variable neighborhood search (VNS) to solve the multicriteria shortest path in multimodal public transportation. The algorithm considers total travel time, travel cost, and number of transfers as criteria. Four modes of transportation are considered: PBN, train, tramway and walking. The initial population for GA is generated using a heuristic shortest path algorithm. For each OD pair, two opposing paths are generated, which are then combined into one path using a labeling technique based on visited nodes. VNS is used to enhance the initial population so that there is a better possibility of finding an optimal solution. After VNS, each individual path in the population is evaluated using the weighted average ranking formula. If the desired criteria are met, the algorithm is stopped, otherwise the usual GA steps in the mating pool are performed, from selection to mutation. Then the process is repeated until the criteria to stop are met. The tests were performed on the dataset of the city of Paris, France, with 17950 nodes (Dib et al., 2017).

In a follow-up work, (Dib et al., 2018) they presented another algorithm to find the shortest multicriteria path in UMTN. The same steps as in the previous work with Hill Climbing Algorithm (HL) instead of VNS. Moreover, the travel time in the transfer edges is considered as time varying.

(Dalkılıç et al., 2017) presented a web application that helps passengers in the city of Izmir, Turkey. It find the shortest route that meets multiple criteria. The platform has a backend that updates schedules every day. It also collects traffic data from various operators and translate it into the General Transit Feed Specification (GTFS), which are later combined into one infrastructure. Stations with a high number of pedestrian connections are called transit stations and are used as accessibility nodes. The main algorithm uses KSP to find incremental paths with increasing number of transfers. For example, it starts searching for k paths with zero transfers, if there are none, it increases the number of transfers, and the algorithm terminates when it finds them. In this way, the passenger insures a trip with as few transfers as possible.

In (López & Lozano, 2020), a shortest hyperpath algorithm was presented to solve the shortest path problem in UMTN. Considering that in some emerging cities, transit schedules are not fully adhered to and only some lines arrive on time, the authors included random arrival times in their model to create a realistic scenario. The methodology uses a label correction algorithm to find the shortest hyperpath based on several criteria. The criteria include maximum number of transfers, maximum walking time, preferred modes, etc. The algorithm was tested with the dataset of Southern Area of Mexico City, Mexico, with 39871 nodes.

In (Peng et al., 2021), a metaheuristic algorithm was presented to find a multi-objective shortest path in UMTN. The algorithm is a mixture of GA and Monte Carlo simulation (MC). Each path is represented by two chromosomes. In the first, each gene is a node of the path, and in the second, each gene is a line edge. The paths are calculated using the Floyd-Warshall algorithm for the shortest path. The modes of the path segment represented by the second type of chromosome are randomly selected. This set of pathways represents the GA initial population. Then, the fittest chromosomes are selected based only on travel time and undergo crossover and mutation. Two fitness approaches are presented, one for a single population and the

other for multiple populations, both based on MC. The algorithm was tested with a simulated dataset of 30 nodes.

2.3 Multimodal Public Transportation Networks Design Problem in UMTN

Since the majority of public transportation users are not owners of motorized vehicles, this research area focuses on reconfiguring public transportation-only networks to enable better interaction. An efficient MPTN design considers the interaction of different transportation modes. A new trend in MPTN design is the use of a hub-and-spoke network topology. In this trend, the hubs are located in rail-based transportation networks and the bus lines are used as spoke lines and allocated to the hubs. This model simplifies the problem to some extent and can support the development of feasible solutions. Furthermore, literature reports that a trip on a well-developed hub-and-spoke network can compete with private transportation (Daganzo, 2010). In this way, more private car users are attracted to public transit.

In a hub-and-spoke network infrastructure, hub facilities aggregate flows from different origins and distribute them to their intended destinations. The main advantages of this infrastructure are the use of fewer edges to connect a large number of origin-destination pairs (OD) and the exploitation of economies of scale by concentrating traffic flow on the hub-level network. An example of a hub-and-spoke network is shown in Figure 2.2. Compared to a fully connected network, 5 fewer edges are used in this topology, making optimal use of resources. Hub networks are categorized based on the number of hubs in the network. For example, the hub network shown in Figure 2.2. is a single-hub network. Multiple-hub networks are shown in Figure 2.3. This category can be further subdivided by the way the spokes are allocated to the hubs. In Figure 2.3.a, the single-allocation is shown, where each spoke is connected to only one hub. Figure 2.3.b shows the multiple assignment, where the spokes can be assigned to multiple hubs, and finally the flexible-multi-allocation is shown in Figure 2.3.c, where connections between the spokes (ISC) are allowed. Directing traffic flow on fewer lines makes the network easier to manage

and reduces maintenance costs, which are part of the economies of scale. An important task in building a hub-and-spoke network is locating the hub facilities. In literature, this task is known as the Hub Location Problem (HLP). For more details related to hub-and-spoke infrastructure and HLP, readers are referred to (Alumur & Kara, 2008; Farahani et al., 2013).

The first work in the field of HLP was by O. Kelly, who developed a quadratic integer formulation to minimize total transport costs. There, the hub network is fully connected, which means that each trip is routed through at most two hubs. A discount is given if the traffic flow is routed through the hub edges (O'kelly, 1987). Subsequently, in the last three decades, HLP has been considered in many research areas, such as air transport, freight transport and telecommunications. Unlike to these fields, MPTN has not received much attention in the literature.

Figure 2.4 shows an application example of hub-and-spoke infrastructure in MPTN. Here, the subway stations are the hubs and the bus lines are the spokes. This is the general architecture used in the literature. The subway stations are used as hubs because of their high capacity and speed. The bus network is used as spokes because of its flexibility.

In the first HLP model applicable to MPTN (Nickel et al., 2001), a mixed-integer formulation was presented. The condition that the hub network must be fully connected was relaxed because it was not applicable in public transport. There, the hubs were located in the subway network and the spokes were in the bus network. The proposed method was tested using the CAB dataset (O'kelly, 1987). In a related work, (Gelareh & Nickel, 2008), a Bender decomposition approach was proposed to solve the HLP in MPTN. The method was tested with the AP dataset (Ernst & Krishnamoorthy, 1996).

In (Sung & Jin, 2001) an integer programming method was used to solve the HLP with the aim of minimizing costs. The method divides the network into clusters and assumes that the number of clusters is known in advance. In the experiments, datasets with 50 nodes are used for clustering.

In (J. Yu et al., 2008), a cluster-based approach to HLP was presented. The researchers remodeled the problem into an integer nonlinear programming formula and obtained a global optimal solution for the objective of minimizing total travel time. The proposed formulation contains a small number of variables, making it suitable for solving with mixed integer formulation tools. The program was solved using the solver LINGO MIP. In a follow-up work, (J. Yu et al., 2009), a cluster-based hierarchical approach for HLP was presented. In this Thesis, the authors linearized their earlier formulation to improve the computation time. (J. Yu et al., 2011) presented a multicriteria decision model based on analytical hierarchy processes and fuzzy logic. This model evaluates hub plans based on six criteria: Overall network efficiency (optimal travel time), transfer volume, proximity to high demand areas (airport, transit station, etc.), uniformity of hub distribution, availability of land for construction, and construction cost. A recent study (Yuan & Yu, 2018) presented a hybrid meta-heuristic using genetic and augmented Lagrangian dual algorithms to solve HLP. The overall performance of the system was given as the optimization objective. A vector of variables was used to represent the chromosomes of the GA. The vector contains decisions about clusters and whether a node is a hub or not.

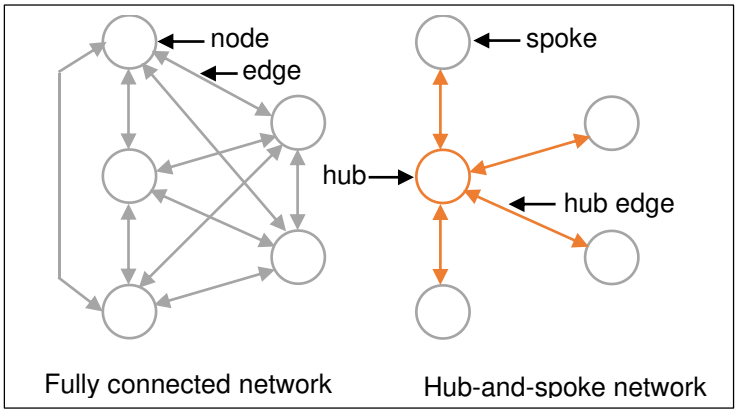


Figure 2.2. Demonstration of the number of edges used in a hub-and spoke network to connect the same number of nodes as a fully connected network.

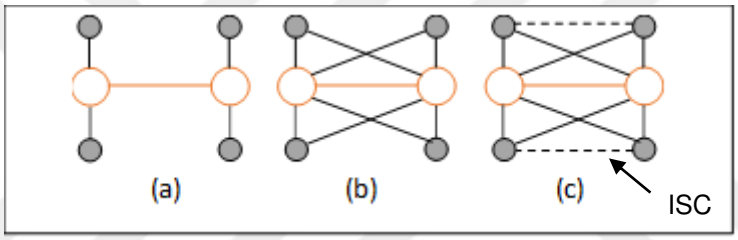


Figure 2.3. Different types of multi-hub networks: (a) is a single-allocation hub network, (b) is a multi-allocation hub network and (c) is also a multi-allocation hub network with inter spoke connections (ISC).

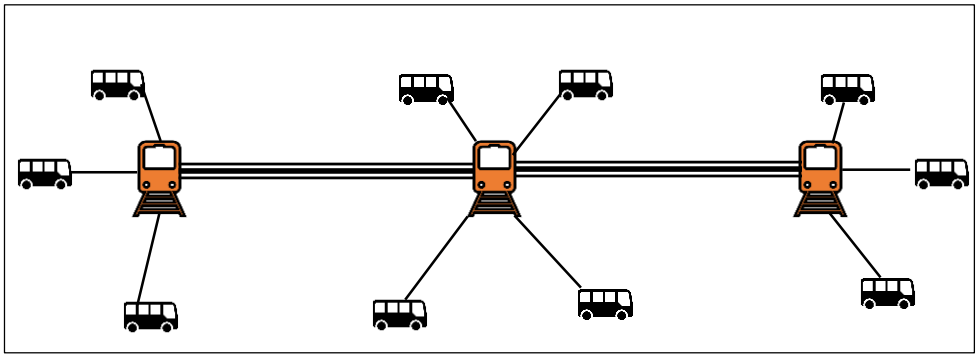


Figure 2.4. Example application of a Hub-and-Spoke topology to public transportation.

The initial population of the GA was generated using the CIPLEX. The implemented fitness function ensures that the solutions are compatible with the constraints given in the formulation. One-cut crossover and random selection were used to generate the offspring.

In (J. Yu et al., 2008, 2009, 2011; Yuan & Yu, 2018) the network of Suzhou Industrial Park, China was divided into zones, were each zone represented by a node. The nodes were clustered based on geographical and administrative aspects. It was also assumed that the number of clusters and hubs are known in advance. In these works, the study area was divided into 58 zones and centroids of these zones were subjected to clustering.

In (Z. Wang & Chen, 2012), a single-objective optimization procedure was presented to locate hubs in a hierarchical hub structure. A trip volume criterion was used to divide the network into zones. The clusters (zones) were assigned different predefined radius values depending on the hierarchy level. The study area was grouped into 12 zones. The procedure generates PBN, express bus, and feeder lines according to the location of the hubs.

In (B. Yu et al., 2013), a bi-level approach to find optimal hub location was developed. At the lower level, candidate hubs were selected based on demand magnitude in a given radius, and then at the upper level, final hub locations were selected by calculating the cost of a hub per demand covered. This approach was applied to the public transportation network of Dalian, China. However, due to the large size of the dataset, the authors had to aggregate nodes and the dataset was reduced from over 3000 nodes to 998.

In (Majima et al., 2017) a methodology based on the growing network model (Gastner & Newman, 2006) is presented. A genetic algorithm is used to locate a single hub, which is considered as the root of the network. Then, according to the principles of the growing network, bus lines are connected to the hub. The chromosomes of the GA represent the longitude and latitude of a station. The fitness function considers the cost of the operator and the user, i.e., the number of vehicle

and the travel time, respectively. In addition to GA, the Cuckoo Search Algorithm (CS) was implemented to solve the HLP and compare the results with GA. To make a fair comparison, the initial nest number of CS was set to the same initial population number of GA. The proposed methods were tested on a 50-node dataset from Mitaka, Japan. It was found that CS outperforms the computation time of GA.

(Huang et al., 2018) is the only study that used a clustering algorithm to locate the hubs. They used Clustering by Fast Search algorithm presented in (Rodriguez & Laio, 2014). The algorithm finds clusters in two phases. In the first phase, it calculates the local density (ld) based on a radius parameter and the greater density (gd), which is the distance to the other nodes that have a higher local density. In the second phase, the cluster centers are determined by drawing the decision graph and finding the appropriate weights for ld and gd. Nodes located in the upper right corner of the decision graph are called cluster centers, and each node located in the radius of a cluster center is assigned to that cluster. The algorithm finds the clusters automatically, but determining an appropriate radius value for ld is complex and requires prior knowledge of the dataset (Estivill-Castro & Lee, 2000). The algorithm was tested on a large dataset (city of Nanjing, China). Similar to the work cited above, the study area was divided into grid-like zones and the clustering algorithm was applied to each zone separately. However, the number of nodes used in clustering the zones was not reported.

3 CONTRIBUTION OF THIS THESIS

Previous work shows that clustering is an important aspect of locating hubs in MPTN designs. The quality of solutions at all levels of MPTN design depends on the effectiveness of clustering algorithms. However, the following problems related to clustering methods stand out in the referenced works:

- Most studies assume and use predefined cluster zones to locate hubs.
- They require the number of clusters as input (except (Huang et al., 2018)).
- Reductions in dataset size are made to reduce computation time.
- It is often not possible to estimate the performance of the algorithm on other networks because most clustering solutions are problem-specific and also require an initial step to set the parameters for the test network.
- Most of the studies tested on the HLP benchmarks used in operations research such as CAB, AP and TR81. However, MPTN datasets are relatively large compared to those benchmarks. For example, the city of Sydney in Australia has an MPTN with 24063 nodes, where each node is a stop of a transportation mode (Kujala et al., 2018).
- Previous studies do not consider finding hub lines. An exception is the work of (Yuan & Yu, 2018).
- Locating hubs on rail-based networks ignores large portions of the city. However, only developed areas with high demand can benefit from a hub network designed in that manner.

This thesis aims to solve these problems in two phases. In the first phase, an algorithm to find clusters is presented. This algorithm does not require the number of clusters or predefined cluster zones as input. It can handle any network without

simplifying the dataset. In addition to these advantages, the clusters generated by the algorithm are evenly distributed across the networks and have feasible sizes.

In the second phase, a heuristic algorithm for finding hubs and hub lines is presented. According to (B. Yu et al., 2013), in MPTN design, it is more favorable to place multiple hubs and maximize the use of resources, as this provides better accessibility for passengers and more choices when transferring. In addition, a larger coverage of the network generates more revenue. Therefore, the presented algorithm focuses on maximizing the network coverage. Finally, a bus network-based hub network is created in the area without rail lines.

The performance of the resulting network is tested using the Greater London MPTN and compared with the Journey API provided by Transportation for London (TFL).

4 BACKGROUND

4.1 Spatial Clustering

The proposed algorithm makes use of the spatial data mining methods. In the following, a brief discussion of spatial data mining is given to familiarize the reader with the concepts and methods used in this thesis.

Spatial data mining is the process of extracting meaningful features, important information and knowledge from spatial datasets (Ng & Han, 1994). Spatial clustering is the backbone process of spatial data mining; similar data points are grouped into the clusters. In general, similarity is measured based on the Euclidean distance between data points (Xu & Tian, 2015). Accepted categories of clustering algorithms and some of the prominent ones in each category are as follows:

- **Partitioning** (*k-means* (MacQueen & others, 1967), *k-medoids* (Park & Jun, 2009) and clarans (Ng & Jiawei Han, 2002))
- **Hierarchical** (*chameleon* (Karypis et al., 1999), *rock* (Guha et al., 2000) and *brich* (Zhang et al., 1996)).
- **Density-based** (*dbscan* (Ester et al., 1996), *optics* (Ankerst et al., 1999) and *denclue* (Hinneburg et al., 1998))
- **Graph theory-based** (*autoclust*, *acdt* (Deng et al., 2011) and *triclust* (D. Liu et al., 2008))
- **Grid-based** (*clique* (Agrawal et al., 1998), *sting* (W. Wang et al., 1997) and *wavecluster* (Sheikholeslami et al., 1998)).
- **Model-based** (*coweb* (Fisher, 1987), *art* (Carpenter & Grossberg, 1987) and *gmm* (Rasmussen, 2000)).
- **Artificial intelligence** (*ant* (Handl & Meyer, 2007), *van* (der Merwe & Engelbrecht, 2003)), and others.

Clustering algorithms have proven to be effective for various application domains and can overcome several clustering challenges. For a comprehensive review of clustering algorithms, readers are referred to (Xu & Tian, 2015). Some of the recent spatial clustering algorithms use Delaunay triangulation in determining the relation of points (*autoclust* (Estivill-Castro & Lee, 2000) *acdt* (Zhan et al., 2017), and *triclust* (D. Liu et al., 2008)). These algorithms can identify complex clusters by using global and local statistical features of the dataset. For example, *autoclust* is able to identify clusters with different densities, while *acdt* and *triclust* can identify clusters with different internal densities.

4.1.1 DBSCAN: Density-based Spatial Clustering of Applications with Noise

Dbscan is a clustering algorithm that considers the minimum number of neighbors (*minPts*) that a spatial point has in a predefined radius (*esp*) to decide whether to group the points in the same cluster or not. The spatial points are divided into three types: Core, Border and Noise. Everything depends on the reachability of the points. If the neighborhood density of a point is greater than or equal to *minPts*, it is a core point; if a point is accessible from a core point and has a density less than *minPts*, it is a border point; otherwise, the point is a noise point.

Figure 4.1. shows an example of *dbscan's* clustering process. In this figure, there are two clusters colored orange and light gray. The point A is a core point that belongs to the gray cluster. The points colored in blue also belong to the clusters. They are called border points because they have a core point in their neighborhood, but they also have fewer points than *minPts*. For example, point B has 3 points in its neighborhood (including B), one of which is a core point and the other is a noise point. A noise point is not accessible from any core point. Point C, for example, has 3 points in its area, but none of them is a core point, so it is called a noise. The pseudocode of *dbscan* is shown in Algorithm 1.

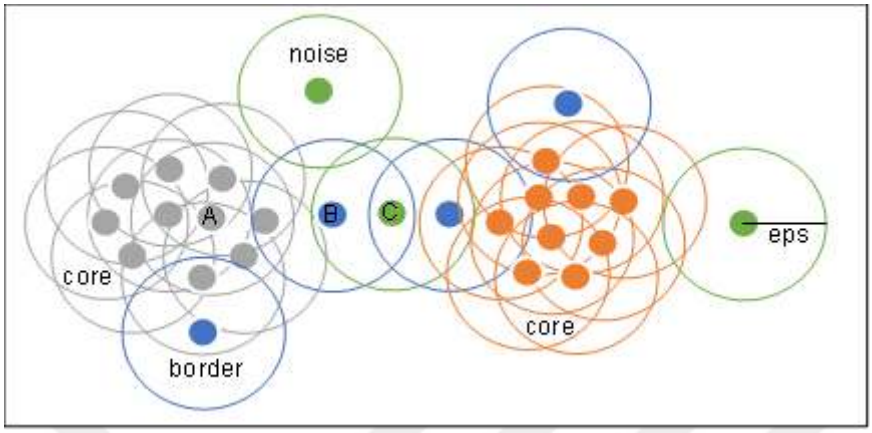


Figure 4.1. An example of dbscan clustering process with minPts =4.

At the start of *dbscan*, if a point *x* satisfies the condition of *minPts*, a cluster *C* is created and the point is added to it. Then, if a neighbor of *x*, in this case *k*, is also a core point, all neighbors of *k* are added to the cluster *C*, which is known as a chain of density. However, by using a static value for *eps*, *dbscan* is able to identify only clusters with similar density. It is insufficient when searching for clusters with different densities.

Algorithm 1: Density-based Spatial Clustering of Applications with Noise Adapted from (Schubert et al., 2017)

Input: The set of n spatial points SP, the radius esp, the minimum number of points in a cluster minPts.
Output: set of clusters

```
for i ← 1 to n do
    if spi is not visited then
        setVisited(spi)
        listOfNeighborsi ← getNeighbors(esp, spi)
        if | listOfNeighborsi | < minPts then
            setAsNoise(spi)
        else if spi is a core point then
            C ← createCluster
            C ← spi
            for x ← 1 to | listOfNeighborsi | do
                if spx is not visited then
                    setVisited(spx)
                    listOfNeighborsx ← getNeighbors(esp, spx)
                    if | listOfNeighborsx | >= minPts then
                        C ← spx
                        listOfNeighborsi ← listOfNeighborsi U listOfNeighborsx
                    end
                end
            end
        end
    end
end
```

4.1.2 OPTICS: Ordering points to identify the clustering structure

Optics is inspired by *dbscan*, it also identifies clusters based on point density. However, it is able to identify clusters with different densities. It also uses a radius *eps* and the minimum number of points *minPts*. However, as the name suggests, optics sorts the points according to their accessibility-distance (also known as reachability-distance). To calculate the accessibility-distance, the core-distance of a point must be determined. The core-distance is the minimum distance at which the

point can still have *minPts*. If a point is at a distance less than or equal to the core-distance, its accessibility-distance is equal to the core-distance. If a point is at a distance greater than the core distance, its accessibility-distance is the Euclidean distance (or some other metric used in the scope). After the reachability-distance has been calculated for all points, the points are ordered according to their reachability, for more information see (Ankerst et al., 1999). Figure 4.2. shows an example of the reachability-distance process of the optics algorithm, adopted from the original paper. The point *x* is a core point. Its core-distance is *4 mm*, which is the minimum distance to be classified as a core point. Its accessibility-distance is *6 mm* to the point *k*.

Figure 4.3. shows a demonstration of cluster detection using the reachability-plot. The clusters are found manually by vision, i.e., a person must observe the plot to detect the "valleys". A valley is the steepest change in the plot. At *optics*, the steeper the valley, the denser the cluster. The example shown in Figure 4.3 was created using the ELKI (Environment for Developing KDD-Applications Supported by Index-Structures) tool (Erich & Arthur, 2019). The "Vary Density"

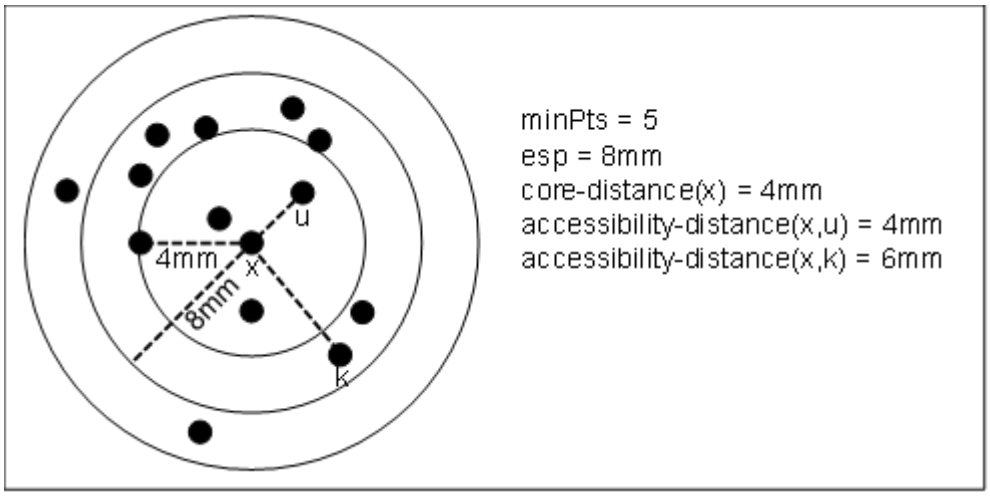


Figure 4.2. Example of calculating the reachability-distance in optics.

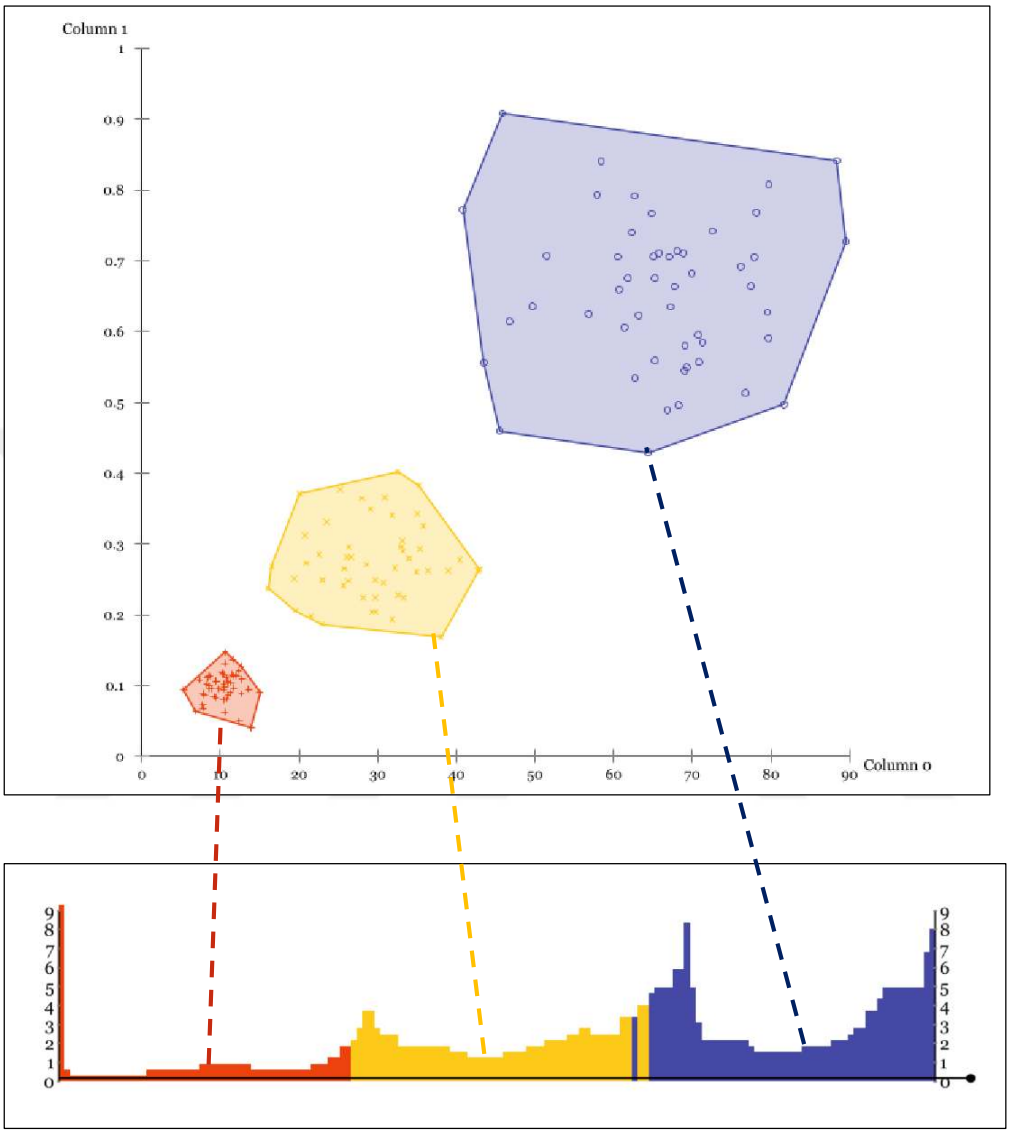


Figure 4.3. A demonstration of the reachability-plot of optics using the ELKI tool with the Vary Density dataset. minPts = 10 and eps =10.

dataset (https://elki-project.github.io/datasets/) with 150 spatial points was used. It contains 3 clusters with different densities, which is a difficult task for density-based clustering algorithms. As you can see from the reachability-plot, the orange cluster has a steeper slope compared to the others. Despite the different densities, *optics* easily detects these clusters, which is impossible for *dbscan*. The pseudocode can be found Algorithm 2.

Algorithm 2: Ordering points to identify the clustering structure, adapted from (Ankerst et al., 1999)

Input: The set of n spatial points SP, the radius esp, the minimum number of points in a cluster minPts. The list containing the ordered points oL.
Output: ordered list of points

```
for i ← 1 to n do
    if spi is not visited then
        listOfNeighborsi ← getNeighbors(esp, spi)
        setVisited(spi)
        setReachabilityDistance(spi, "undefined  ")
        core-distance(esp, minPts, spi)
        addedPoint(oL, spi)
        if core-distance(spi) is "undefined" then
            updatePoint(spi)
            while oL is not emply do
                spx ← oL.getNext()
                listOfNeighborsx ← getNeighbors(esp, spx)
                setVisited(spx)
                core-distance(esp, minPts, spx)
                addedPoint(oL, spx)
                    if core-distance(spx) is "undefined" then
                        updatePoint(spx)
                    end
            end
        end
    end
end
```

4.1.3 AUTOCLUST: Automatic Clustering

As it can be noticed, *dbscan* and *optics* require two parameters *esp* and *minPts*. These parameters can affect the result of the algorithm and it can be very difficult to set appropriate values for them. *Autoclust*, on the other hand, can identify clusters with different densities without any parameter.

Autoclust uses Delaunay triangulation (DT) to detect the proximity between spatial data points (DT is explained in section 4.1.4). The algorithm finds clusters in three steps. In the first step, the algorithm classifies DT edges using three thresholds calculated based on statistical data for each data point. The first threshold extracts very long edges (VLE), which are links between clusters. The second threshold extracts very short edges (VSE) that can form linear bridges (noise), which also affects good edges within clusters. The third threshold extracts the other edges (OE) whose statistical data satisfy the local and global conditions. The edges that belong to VLE and VSE are removed. However, some edges may also belong to VSE and OE, so removing them may disconnect the interior of some clusters. Therefore, the second step of the algorithm is to recover these edges by calculating the statistical data of each point. The last step consists of removing bridges and noise. For a detailed explanation of the statistical information extracted from DT, the readers are referred to (Estivill-Castro & Lee, 2000). Like *optics*, *autoclust* can identify clusters

with different densities, though it can also find clusters that are nested within other clusters. The pseudocode of *autoclust* can be found in Algorithm 3.

Algorithm 3:Automatic Clustering

Input: the set of spatial points SP.
Output: clusters.
DT ← computeDelaunay(SP)
for sp ← 1 to | DT_{points} | **do**
  $Mean_{sp}$ ← calculateLocalAvg(sp_{edges}).
  $StandDev_{sp}$ ← calculateLocalStdev(sp_{edges}).
  $Mean_{all}$ ← calculateGlobalAvg(DT_{edges}).
  $StandDev_{all}$ ← calculateGlobalStdev(DT_{edges}).
  $MeanStdev_{sp}$ = sumOfAllLocalStdev(DT)
  $RelStdev_{sp}$ = $StandDev_{sp}$ / $MeanStdev_{sp}$
  if |$neighborEdge_{sp}$| < $Mean_{sp}$ - $StandDev_{all}$ **then**
    classifyEdge(VSE_{sp}, $neighborEdge_{sp}$)
  if |$neighborEdge_{sp}$| < $Mean_{sp}$ + $StandDev_{all}$ **then**
    classifyEdge(VLE_{sp}, $neighborEdge_{sp}$)
  if sp_{edges} ∈ VSE U VLE **then**
    classifyEdge(OE_{sp}, sp_{edges})
  end
  deleteEdges(VSE)
  deleteEdges(VLE)
  recoverEdges(VSE, OE)
  deleteBridges()
end

4.1.4 Delaunay Triangulation

In computational geometry, any set of vertices in a plane can be triangulated as long as there are no intersections between the edges and the vertices do not lie on the same line (Lee & Schachter, 1980). Delaunay Triangulation (DT) was introduced by Boris Nikolaevich Delaunay (Delaunay, 1934) . Triangulation is performed so that the inside of the circumcircle of a triangle is empty. A circumcircle is a circle

that passes through all vertices of a triangle. If the circumcircle of a triangle contains other vertices that do not belong to the triangle, the triangle is not a valid Delaunay triangle. The circumcircle technique ensures that the minimum angle in the triangulation is maximized, which means that only points that are a short distance apart are connected by an edge. A given set of points can be triangulated in various ways, but will have only one DT, unless the points form a rectangle that has two valid DT ("Computational Geometry," 2008) (Chapter 9).

Several algorithms for computing a DT are presented in the literature (FORTUNE, 1995; Su & Scot Drysdale, 1997). The three most commonly used algorithms for triangulating 2D vertices in a plane are the following: *Incremental* (Guibas et al., 1992) , *Divide and Conquer* (Lee & Schachter, 1980) , and *Sweephull* (Sinclair, 2016). In this Thesis, the incremental algorithm is used, more specifically, the Random Insertion Incremental Delaunay Triangulation (RIIDT) algorithm. The reason for this choice is that this algorithm is straightforward and easy to implement. Even though the other algorithms are considered faster, the speed becomes almost the same for large data sets.

Algorithm 4 shows the pseudocode of RIIDT. The first step of RIIDT is to create a temporary triangle large enough to encompass all vertices of the data set. At this point, this triangle represents a DT with one triangle. In the second step, the algorithm randomly inserts vertices. Only one vertex is randomly selected and added to the current triangulation. There are two possible positions for a newly inserted vertex: inside a triangle or on an edge. If the vertex is inside a triangle, the triangle is split into three triangles by creating an edge between the inserted vertex and the vertices of the initial triangle, then the three edges are checked for validity. If the vertex is on an edge that belongs to two triangles, then each triangle is split into two triangles by creating an edge from the inserted vertex to the vertex's opposite edge. Then all four edges connected to the new vertex are checked for validity.

Algorithm 5 shows the pseudocode of the algorithm ValidateEdge, which is a sub algorithm of RIIDT that checks if an edge is a valid Delaunay edge. An edge

is said to be a valid Delaunay edge if the associated circumcircle does not contain any vertex other than the vertices of the same triangle. If an edge is invalid, it is removed and replaced, this operation is known as the flip. This algorithm is recursive. A flipped edge is checked again for validity until no more flips are required.

Algorithm 4: Random Insertion Incremental Delaunay Triangulation. Adapted from ("Computational Geometry," 2008)

Input: The set of n vertices V in the Euclidean space
Output: The Delaunay Triangulation DT of V
Initialization;
Scan all vertices in V and generate a DT composed of one large triangle T ($v_a v_b v_c$) which contains all the pints in V;
for i ← 1 to n **do**
    add random vertex v_i to DT;
    detect a triangle $v_p v_q v_r \in$ DT that contains v_i;
    If v_i is located inside of the triangle $v_p v_q v_r$ **then**
        divide the triangle $v_p v_q v_r$ to three triangles by adding edges from v_i to the vertices of $v_p v_q v_r$;
        ValidateEdge(v_i, $\overline{v_p v_q}$, DT);
        ValidateEdge(v_i, $\overline{v_q v_r}$, DT);
        ValidateEdge(v_i, $\overline{v_r v_p}$, DT);
    else
        (In this case v_i is located on one of the edges of $v_p v_q v_r$, for instance let's take $\overline{v_p v_q}$);
        create edges from v_i to v_r and to the vertex v_x that belongs to the triangle incident to $\overline{v_p v_q}$, thus, the two triangles that shares the edge $\overline{v_p v_q}$ are divided to four triangles;
        ValidateEdge(v_i, $\overline{v_p v_x}$, DT);
        ValidateEdge(v_i, $\overline{v_x v_q}$, DT);
        ValidateEdge(v_i, $\overline{v_q v_r}$, DT);
        ValidateEdge(v_i, $\overline{v_r v_p}$, DT);
    end
end

Algorithm 5: ValidateEdge.

Input: vi is the vertex added, vrvp is the edge of DT that might be flipped

If $\overline{v_r v_p}$ is invalid then
  get vrvpvq which is the triangle that shares the edge $\overline{v_r v_p}$ with
  the triangle vivrvp;
  remove edge $\overline{v_r v_p}$;
  add edge $\overline{v_q v_i}$;
  ValidateEdge(vi, $\overline{v_i v_r}$, DT);
  ValidateEdge(vi, $\overline{v_r v_q}$, DT);
end

5 TEST DATASETS GENERATION AND SIMULATION

As it is shown in Table 2.1, most of the previous work were tested on very small data sets. Contrary, the datasets used in multicriteria shortest path are relatively larger than the datasets used in Table 2.1. Our goal In this Thesis is to create a realistic scenario for MPTNs. Therefore, the use of a real network is necessary to demonstrate the performance of the proposed method.

An important step in the creation of data sets is the visualization of the data set. It is necessary to confirm the result of the algorithm and even helps in the design phase of the algorithm.

Real city network data is often difficult to obtain. Researchers tend to extract geographic data from *Google Maps* or *Open Street Maps* (Haklay & Weber, 2008), which is time consuming and error prone. Moreover, these benchmarks are provided in various formats such as databases, scripts, algorithms, classes, etc., which makes it difficult for researchers to adopt them in their implementations. In a recent work, Ahmed et al. pointed out the lack of available benchmarks for the routing problem and question the reliability of the performance results based on small networks (Ahmed et al., 2019). It is worth noting that the problem is not only the size of the network, but also the shape of the network is usually oversimplified.

The urban transportation network can be modeled as a graph. There are several high-quality graphing tools that can be divided into two categories (1) for teaching and (2) for graph visualization and analysis. Educational programs such as *CABRI-Graph* (Carbonneaux et al., 1996) and *GraphTea* (Rostami et al., 2014) are intended for teaching graph theory and are not particularly well suited for creating benchmarks for MPTN. Graph visualization and analysis software is well suited for evaluating an existing dataset. They can generate a variety of network representation structures for the given data, such as *Gephi* (Bastian et al., 2009) and *Graphviz* (Ellson et al., 2002).

Libraries for higher-level programming languages can also be used to model graphs. For example, the Python library *NetworkX* (Hagberg et al., 2008) and the Java library *JGraphT* (Michail et al., 2020) provide a variety of commonly used functions and structures for graph processing. Of course, the use of these libraries requires knowledge of the corresponding programming language.

A general review of the relevant literature shows that although graph software and libraries provide sophisticated functions and algorithms, they lack functions for multimodal graph generation and demand data generation. In addition, researchers must contend with different software and libraries to obtain benchmarks that are constrained by size and structure. At the time of writing this thesis, there is no benchmarking tool for multimodal urban transportation in the literature.

In this chapter, I present a tool to generate benchmarks for multimodal urban transport research (MUTBG). As mentioned, there is a real need for such a tool. The tool presented in this thesis can generate benchmarks for a variety of modes: Bus network, private networks, rapid busses, metro, tramway, and inland waterway. The presented software can facilitate the preliminary work of researchers. Different models (single or multimodal networks) can be easily created so that the performance of the planning algorithms can be tested under different scenarios. The tool provides full control over all network parameters; map size, vertices, weights, etc. can be specified through a simple graphical user interface. Benchmark data is output as simple text files, providing flexibility and facilitating benchmark sharing. A simple graphical user interface (GUI) is provided for visualizing and editing the network. The tool can be downloaded as a jar file from the tool's landing page at:

https://urbantransportationbenchmarkgenerationtool.wordpress.com/

5.1 Tool's design and implementation

5.1.1 Description of the options and commands

The main GUI of the tool is shown in Figure 5.1. The GUI displays the main process commands, input options, and parameters. The options and parameters are described below:

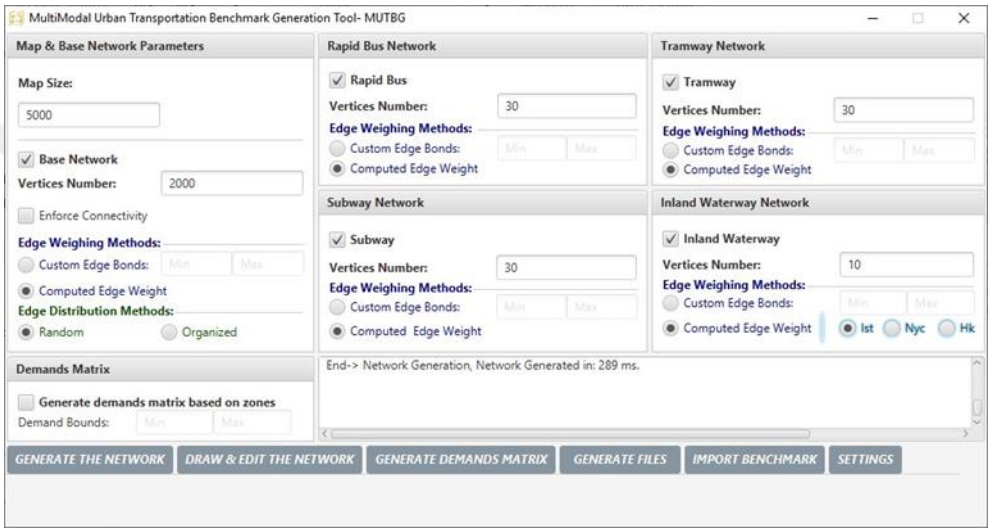


Figure 5.1. Tool's main GUI.

- The map is defined as a square geometric shape. The parameter *Map Size* specifies the length of the border in pixels. The density of the network depends on the map size and the number of vertices. For example, this option allows the user to obtain networks for narrow and large cities. If a network with 300 vertices is created on 800 and 1600 map sizes, the first map will have a denser distribution of the nodes than the second one. In addition, it is possible to manually change the density of any area by editing the network using the drawing GUI.
- You can use the Basic Network, Rapid Buses, Subways, Trams, and Inland Waterways checkboxes to create different benchmarks. A base network can represent a public bus network, a private network, or both.

- All network modes require the following inputs: the number of vertices and the selection of the method to weight the edges. There are two methods to set the weighting: customized and computed.
- The inland waterway network can be created based on the topologies of the cities of *Istanbul*, *New York* and *Hong Kong*. Check boxes are available to select one of these options.
- The edges of the base network are connected using random or organized methods. The details of these methods are explained in the next section.
- To ensure the connectivity of the network, you can select the 'Enforce connectivity' option in the main GUI. This option is also available in the drawing GUI.

The main process commands are shown below in the GUI in Figure 5.1. The Generate Network process manages the automatic network generation function. Other processes can be used after the network is ready. The Draw and Edit Network process allows visualization of the network, navigation through the map and editing of the network, etc. The Generate Demand Matrix process creates a two-dimensional matrix of demand values between origin and destination pairs. The Generate Files process writes benchmark data to .txt files. In the Settings section you can set new constraints for parameters that require limitation. The aforementioned processes are explained in the following sections.

5.1.2 Software Architecture

Figure 5.2 shows the block diagram for the software architecture using the Javafx MVC (Modal-View-Controller) design pattern. A brief explanation of the execution flow is described over this block diagram as follows: The controller instantiates an empty network and manages the activities between all layers. First, the controller checks the validity of the inputs and then injects them into the model. The basic components of Model are Vertex, Zone, and the Edge classes. The Multi-

modal Network class in Model orchestrates the creation of the network. It notifies the controller when the network is ready. The processing flow continues based on one of the following choices: (1) Display the network (using the Draw GUI), the Draw Controller class controls the display commands. (2) Generate the demand matrix. (3) Write the benchmark data to files.

5.1.3 Network Generation

The flowchart of network generation is shown in Figure 5.3. The first step is the creation of the background map. The map is divided into equal zones. The number of zones is calculated by dividing the map size by 200. The scalar value 200 is chosen for visual efficiency. The zone object contains the following fields: a unique integer id, a type (land or water), a list of neighboring zones, a separate list of vertices for each network mode in the zone, and a rectangle2D field.

Once the background map of the zones has been created, the network for each mode can be placed on it. In the following subsections, all available modes of transport are explained.

5.1.3.1 Base Network

The Base network represents the main transportation infrastructure of an urban area. In this network, an edge represents a road and a vertex represents an intersection. It can be used to create a public bus network and private networks (car, taxi, bike, etc.). The vertices of the base network are randomly distributed on the map. A vertex object is similar in structure to a zone object of the background map. It contains the following fields: a unique id, a transport mode, a list of neighboring vertices, a pointer to its zone, and a Point2D (x, y) field.

Table 5.1: Abbreviations used in the algorithms of this section

Abb.	Explanation
soz	Set of zones
moz	A 2D matrix of zones
bv	Base network vertex
be	Base network edge
sbvcz	Set of base network vertices connected to a zone
sld	Set of local distances, calculated between the current vertex and all vertices allocated in the same zone, object saved in this set contains two fields the destination vertex and, the distance
snd	Set of neighboring distances, similar to sld, except the distances are calculated between the origin vertex in the current zone and the vertices located in the adjacent zones
soad	Set of all distances, it is a concatenation of sld and snd
nbv	Number of neighbors of a vertex
bsoe	Base network set of edges
lc	Set of base network vertices located in the last column in the current zone
lr	Set of base network vertices located in the last row in the current zone
fr	Set of base network vertices located in the first row in the zone below
fc	Set of base network vertices located in the first column in the right zone
fv	Fixed network vertex
fe	Fixed network vertex
flov	Fixed networks list of vertices
floe	Fixed networks list of edges
fl	Fixed networks line, a line is a set of vertices
flol	Fixed networks list of lines
rd	Random direction
iv	Inland waterway vertex
ie	Inland waterway edge
ilov	Inland waterway list of vertices
iloe	Inland waterway list of edges
il	Inland waterway line, a line is a set of vertices
ilol	Inland waterway list of lines
swc	Set of zones located in the west coast
sec	Set of zones located in the east coast
oc	Origin coast
ip	Inland waterway polyline, each polyline is a set of zones that represents the shortest path
ilop	Inland waterway list of polylines

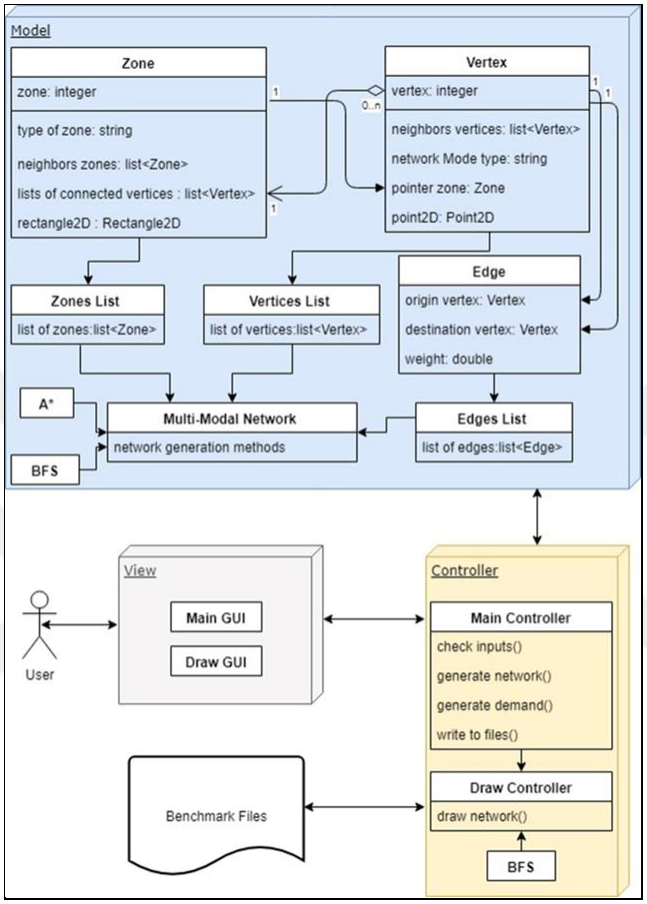


Figure 5.2. Software architecture

The Base network edge object contains the following fields: an origin vertex (Point2D (x1, y1)), a destination vertex (Point2D (x2, y2)), and a weight. The edge weights can be created using customized or computed methods. In the customized method, a random weight is assigned between the minimum and maximum values specified by the user. In the computed method, on the other hand, the Euclidean distance between the starting point and the target point is set as the weight.

Edges can be connected using either the random or the organized method. The steps of the random edge generation algorithm are shown in Algorithm 6. This

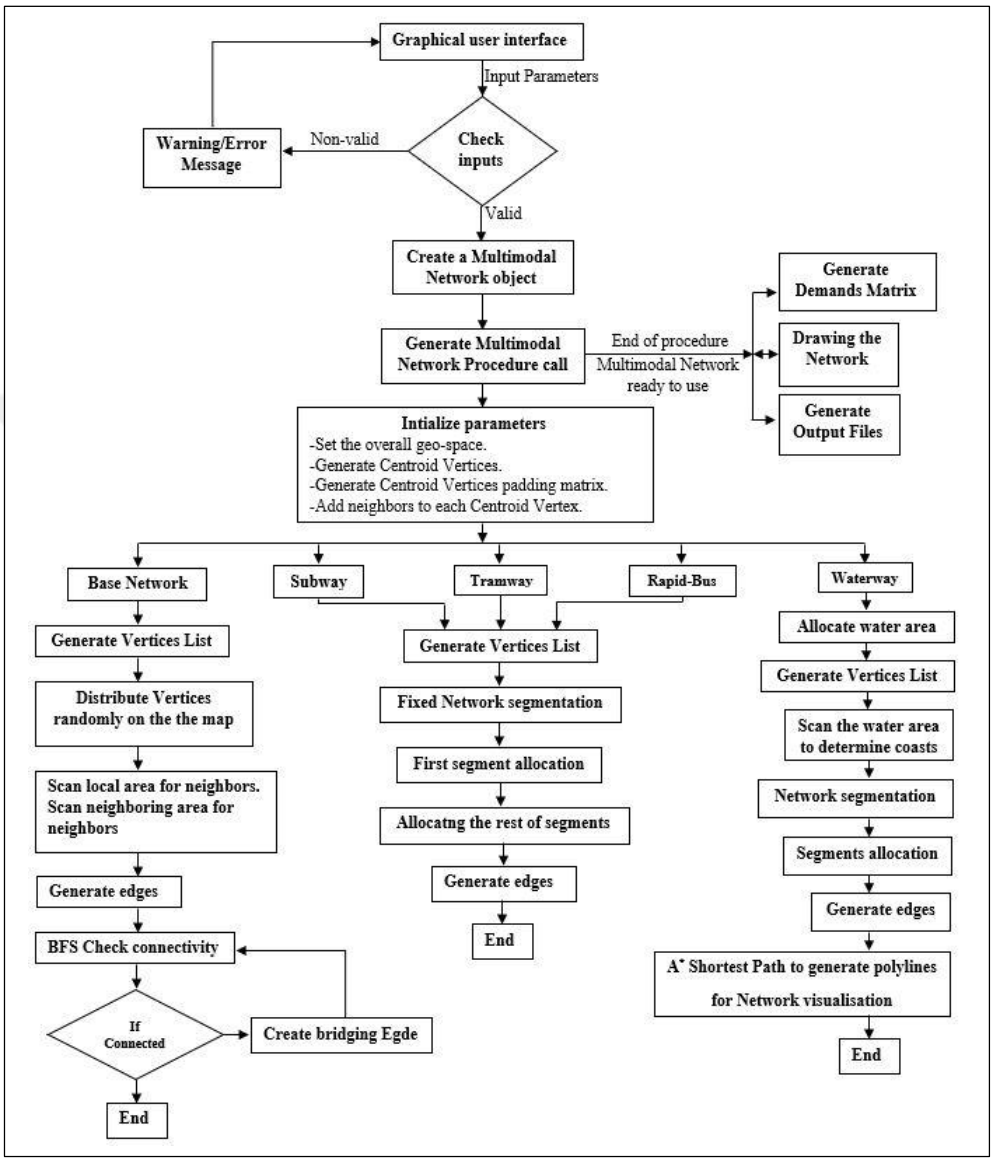


Figure 5.3. The flowchart for Multimodal Network class

algorithm treats the zones sequentially. The distances from a vertex to all other vertices in the current and neighboring zones are calculated and sorted in ascending order. Then the first *nbv* elements of the sorted list become the edges of the vertex,

where *nbv* is a random number between 2 and 7. This default range can be changed in the settings.

The steps of the organized method are shown in Algorithm 7. The purpose of this algorithm is to generate a grid-like network that resembles a modern urban infrastructure. First, the vertices are randomly assigned to zones, then they are aligned. Each vertex is connected to its nearest neighbor on the x and y axes. Edges between adjacent zones are connected in left-to-right and top-to-bottom order.

Algorithm 6: Random edges generation for Base network

```
Inputs: Background map, soz
Output: Random edges.
for each zone in soz do
    for each bv in sbvcz do
        sld ← calculate local distances()
        snd ← calculate neighboring distances()
        soad ← sld ∩ snd
        sort soad ascendingly
        k ← random (2, 7)
        if k <= soad size then
            for i ← 0 until k do
                be ← bv (origin vertex), soad [i] (destination vertex)
                set be weight as customized or computed
                bsoe ← be
            end
        else
            for each bv in soad do
                create edge()
            end
        end
    end
end
```

Vertices in the last column of a zone are connected to vertices in the first column of the zone on the right. Similarly, vertices in the last row are connected to vertices in the first row of the zone below.

5.1.3.2 Enforce Connectivity

As mentioned earlier, vertices are randomly distributed on the map, so adjacent zones are sometimes assigned an insufficient number of vertices. This results in isolated vertices or sub-networks on the map. To ensure connectivity of the base network, the tool provides two options: automatic and manual connectivity enforcement. The automatic method recursively uses the Breadth First Search (BFS) algorithm to check the connectivity of the network. When an unconnected subnetwork is found, it is connected to the rest of the network via an edge. In this way, the connectivity of the network is guaranteed. To avoid memory overload, the automatic method is terminated when the number of vertices traversed is less than the total number of vertices. In this case, the user is prompted to change the network parameters or establish the connections manually. The second method uses the BFS algorithm to support manual connection. The BFS algorithm colors the traversed nodes so that the disconnected parts of the graph stand out. The user can manually connect the non-colored sub-networks to the colored region. It is worth mentioning that when creating an inland waterway network along with the base network, the sea divides the land. Both methods can create an edge that represents an actual suspension bridge. Finally, connectivity problems can occur if the number of neighbors is set by the user to a small range. The range is set to 2 to 7 by default, which reduces the likelihood of this problem occurring. The default value can be changed under Settings.

Algorithm 7: Organized edges generation for base network.

```
Inputs: Background map, soz.
Output: Organized edges.
for each zone in soz do
    for each bv in sbvcz do
        sld ← calculate local distances()
        for i ← 0 to sld size do
            if x position of bv == x position of sld[i] & smallest(distance) then
                be ← bv origin vertex, sld[i] destination vertex
                set be weight as customized or computed
                bsoe ← be
            else if y position of bv = y position of sld[i] & smallest(distance) then
                be ← bv origin vertex, sld[i] destination vertex
                set be weight as customized or computed
                bsoe ← be
            end
        end
    end
end
for each zone in soz do
    lc ← determine last column ()
    lr ← determine last row ()
    fr ← determine first row ()
    fc ← determine first column ()
    connect vertices in the same x position of lc and fc
    connect vertices in the same y position of lr and fr
end
```

```
Algorithm 8: Generate fixed networks
Inputs: map, flov
Output: Fixed networks
flol ← fixed networks segmentation (flov)
for each fl in flol do
    for each fv in fl do
        rd ← random direction()
        while allocated == false do
            if the zone in the chosen rd is feasible then
                allocate vertex
                allocated ← true
            else rd ← random direction() then
                allocated ← false
            end
        end
    end
end
```

5.1.3.3 Fixed Networks

Rapid bus, subway, and tramway networks are considered fixed networks in the tool because they are rarely changed once established. For example, rapid bus service has reserved roads and tramway, subway networks run on rails. Algorithm 8 is used to create the benchmarks for these network modes. The inputs to the algorithm are the background map and the list of vertices (*flov*). The order of the list (*flov*) gives the edge connections. *flov* is randomly split into parts and a list of lines (*flol*) is obtained. The vertices (*fv*) of each line (*fl*) are randomly distributed on the map. A random direction is chosen from north, east, south, west, northeast, southeast, southwest, and northwest. The random direction() procedure checks the backtracking when it is called. To ensure that all lines are connected, random crossing points are set in the network.

5.1.3.4 Inland Waterway network

The inland waterway network (IWT) can be considered a special case of the fixed networks. The main difference is that the number of vertices of a waterway

line is less than in the other fixed network modes. They usually consist of 2 to 5 vertices. The user can choose one of the predefined background map models, which are based on the inland waterway networks of Istanbul, Hong Kong and New York.

Algorithm 9 shows the main steps to create an inland waterway network. First, a network model is selected. As mentioned before, the water maps are predefined, but the exact location of the coasts changes depending on the input parameters. For this reason, the map is scanned for eligible vertices locations,

Algorithm 9: Generate Inland waterway network

```
Inputs: Background map, ilov, ioz
Output: Inland waterway network.
isol ← inland waterway segmentation (ilov)
for each zone in soz do
    if zone belongs to west coast then
        swc ← zone
    else if zone belongs to east coast then
        sec ← zone
    end
end
for each il in isol do
    if oc == 0 then
        allocate the first vertex (sec)
        allocate the remaining vertices (swc)
    else if oc = 1 then
        allocate the first vertex (swc)
        allocate the remaining vertices (sec)
    end
end
for each ie in isoe do
    ip ← A* shortest path (ie, moz)
    ilop ← ip
end
```

where each vertex represents a dock. Similar to fixed networks, the vertices are generated and stored in a list (*ilov*). The list is randomly divided into lines (*isol*). The A* shortest path algorithm is used to connect the vertices of the IWT. It returns a list of zones that represent the shortest path.

5.1.3.5 Network Visualization

The visual version of the network is automatically drawn using Draw GUI (see Figure 5.4). The GUI provides the following functions: Map navigation, add, delete and view details of a vertex or edge, zoom in/out and reset zoom scale, map key description and force connectivity.

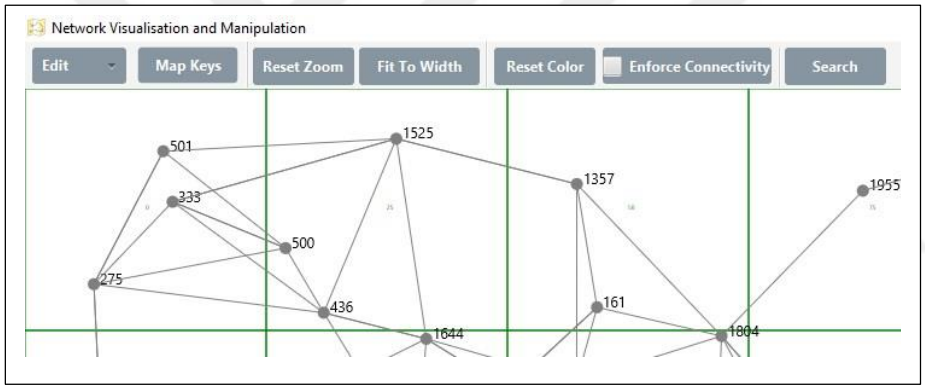


Figure 5.4. Network visualization commands menu

5.1.4 Demand Matrix

The demand matrix can be created only after the network has been generated. The values of the matrix are randomly generated from a uniform distribution, and the minimum and maximum values can be set by the user. Each matrix element represents travel demand between origin and destination zones. The demand matrix can only be created after the network has been generated. The values of the matrix are randomly generated from a uniform distribution, and the minimum and maximum values can be specified by the user. Each matrix element represents travel demand between origin and destination.

5.1.5 Benchmark Files

As mentioned earlier, the network and demand data are written to .txt files. These files are the Zones file, which contains the list of zones including geometric data, the Zones Neighbors file, which contains the list of neighbors for each zone, and the Zones Content file, which contains separate lists of vertices for each network mode in each zone.

For each network mode, the following files are created: the Vertices file, which contains the list of vertices and the zone in which the vertex is located, the Vertices Neighbors file, which contains a list of neighbors for each vertex, the Edges file, which contains the list of edges, including the origin and destination vertices, and the weight of each edge. In addition to these files, for fixed networks, the Lines file is created, which contains a list of lines. And finally, the Demand Matrix file is created, which contains the demand data.

The goal of creating the benchmarks as text files is to facilitate sharing the benchmark instances with other users. In addition, organizing the network structure data into multiple files provides flexibility so that the user can freely choose the components.

5.2 Real World Dataset

5.2.1 Datasets Used in Clustering Experiments

In the experiments of the first phase of the proposed methodology, real spatial datasets of public transport of the cities of Athens, Paris, Sydney, Brisbane, Berlin, Bordeaux, Toulouse, Helsinki, Lisbon and Winnipeg are used (Kujala et al., 2018). These datasets were uploaded to the tool using the "Import Benchmark" function (see Figure 5.1.). The collection of datasets presented by (Kujala et al., 2018) contains 25 MPTN datasets. In this Thesis, in addition to the cities mentioned above, the Rome dataset was used to demonstrate the steps of the clustering algorithm, which can be found at the end of Section 6.1.2. The visualization of these

data sets can be seen in. Figure 5.5. Due to the density of these datasets the zones in the background have been removed for visual clarity.

5.2.2 Dataset Used in Greater London Case Study

To demonstrate the proposed MPTN methods, the MPTN of Greater London, England, is used as a case study. Greater London is divided into 32 boroughs. Most of the public transport network is operated by the government agency TFL, while other parts are operated by private companies. In this thesis, only the network operated by TFL is considered. TFL provides several public APIs that allow access to data and statistics of the transport network (Powered by TFL Open Data. Contains OS data © Crown copyright and database rights 2016, and Geomni UK Map data © and database rights [2019]).

5.2.2.1 TFL Network Data

The network data was extracted using the "Unified API". This API contains several "GET" requests that provide different data about the network. Essentially, three requests were used to obtain the network data. The first request gets all the routes for all the lines in the network based on a parameter called service type. The TFL network has both daytime and nighttime service. The daytime service is called *regular* and the nighttime service is called *night*. In this Thesis, only the regular service is used. The http request used to retrieve this data is as follows:

Request 1: https://api.tfl.gov.uk/Line/**Route**?serviceTypes=**regular**

The response from the server after Request 1 is a JSON array containing all the data about the MPTN routes. From this data the ID of each line is extracted, which in another request is used to extract the stops or stations of each route.

The second query is to obtain the order of stops in each line based on the line ID. This query requires three parameters. The first parameter is the "line id", the second parameter is the "direction", which can be "inbound" or "outbound", and the

last parameter is the "service type". In order to better represent the base network of Greater London, both the inbound and outbound lines for each line are extracted, since in some cases part of the trip may be on different routes. The http query used is shown below, which is an example using line id “h9”, direction “inbound” and service type “regular”:

> Request2: https://api.tfl.gov.uk/Line/**h9**/Route/Sequence/**inbound**?service Types=**regular**

The data received from Request 2 is stored in an object called "Line". This object contains a line ID and two list fields, one list for the inbound route and the other for the outbound route. The lists contain the ordered sequence of stops for each route.

This list is used to create the edges of the network. For each edge created, two weights are set, time and distance. To determine the time it takes to travel between two consecutive stops on a route, a request is made to the Journey API. This API provides the ability to create a multi-criteria query. In this case, the following parameters are used: from (the starting point of the journey), to (the destination of the journey), mode, and time. For the time, a time is chosen that corresponds to the time of day, namely 08:00. The http query used is shown below, which an example showing a query using line “h9” from station “490000101J” to” 490000124HH”.

> Request3:https://api.tfl.gov.uk/Journey/JourneyResults/**490000101J**/to/**490000124HH**?time=**0800**&mode=**bus** HTTP/1.1

The result of Request 3 contains the travel time of the trip. For the distance of an edge, the haversine formula to calculate the geographic distance between two points based on their latitude and longitude is used. The data structure used to create the network is shown in Figure 5.5. The pseudocode of the algorithm used to extract the network from the TFL APIs is shown in Algorithm 10.

The full network includes the following public transport modes: Tube, DlR, TFL Rail, Tram, Overground and Bus. There are 676 bus lines and 15 FPTN lines. All modes of transport together include 31474 stops, of which 30979 are bus stops and 495 are FPTN stops. The bus lines are used to extract the base network. Here, the bus station was subjected to clustering. The clustering algorithm is explained in the next section. The centroids of the clusters are considered as high demand areas and are used to create the base network. First, redundant edges that occur in different lines are removed. If a stop belongs to a cluster, its edge is connected to the corresponding centroid for simplicity. Edges and lines that are not connected to a centroid are discarded and the resulting network is considered the base network. The FPTN and the base network are connected by walking edges to create a complete network. If a stop of the base network is within 500 m of a station of the FPTN, a walking edge is created between them. For simplicity, FPTN stations that serve several modes and are connected underground are grouped together as one station. The complete network is shown in Figure 5.6.

5.2.2.2 TFL Demand Data

The demand data used in this study comes from the 2011 UK Census. More specifically, it is inter-borough demand data in relation to how people travel from their usual place of residence to their place of work (Census output is Crown copyright and is reproduced with the permission of the Controller of HMSO and the Queen's Printer for Scotland. Source: 2011 SWS MSOA/SOA/Intermediate Zone [Location of usual residence and place of work by method of travel to work] - WU03EW – Open). The dataset contains travel statistics from all boroughs of the United Kingdom. Only data for Greater London are extracted. The census provides data on private and public transport; In this Thesis, only public transport is considered. There are three groups; the first group includes: Subway, Metro, Light Rail and Tram; the second group includes: Train, and the third group includes: Bus, minibus or coach. In all districts, 749365 passengers use the first group, 423014

passengers use the second group, and 470837 passengers use the third group. These data refer to networks operated by local authorities and private companies. As the case focuses on TFL operated transport, the demand of the first group is assigned to Tube, DlR, TFL Rail and Tram, the second group to Overground and the third group to bus. For each borough, the total demand of each group is divided equally among the corresponding stations. With the exception of the PBN lines, there are no FPTN lines operated by TFL in some boroughs, so no demand is assigned to the corresponding stations. There are also centroids and FPTN stations located outside of Greater London. No demand is assigned to these stations.

(a) Athens

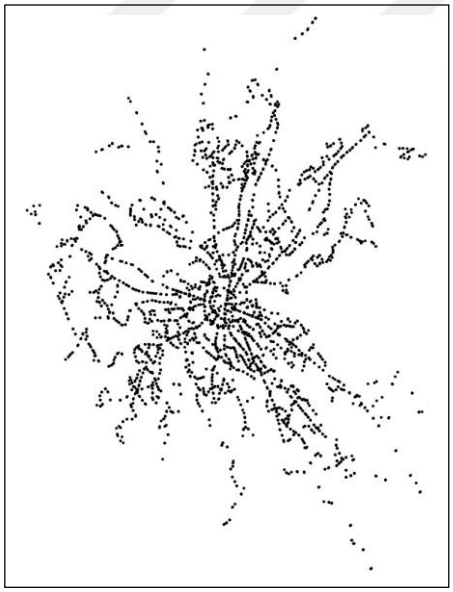
(b) Paris

(c) Sydney

(d) Brisbane

(e) Berlin

(f) Bordeaux



(g) Toulouse

(h) Helsinki

(i) Lisbon (j) Winnipeg

Figure 5.4. The MPTNs used for experiments.

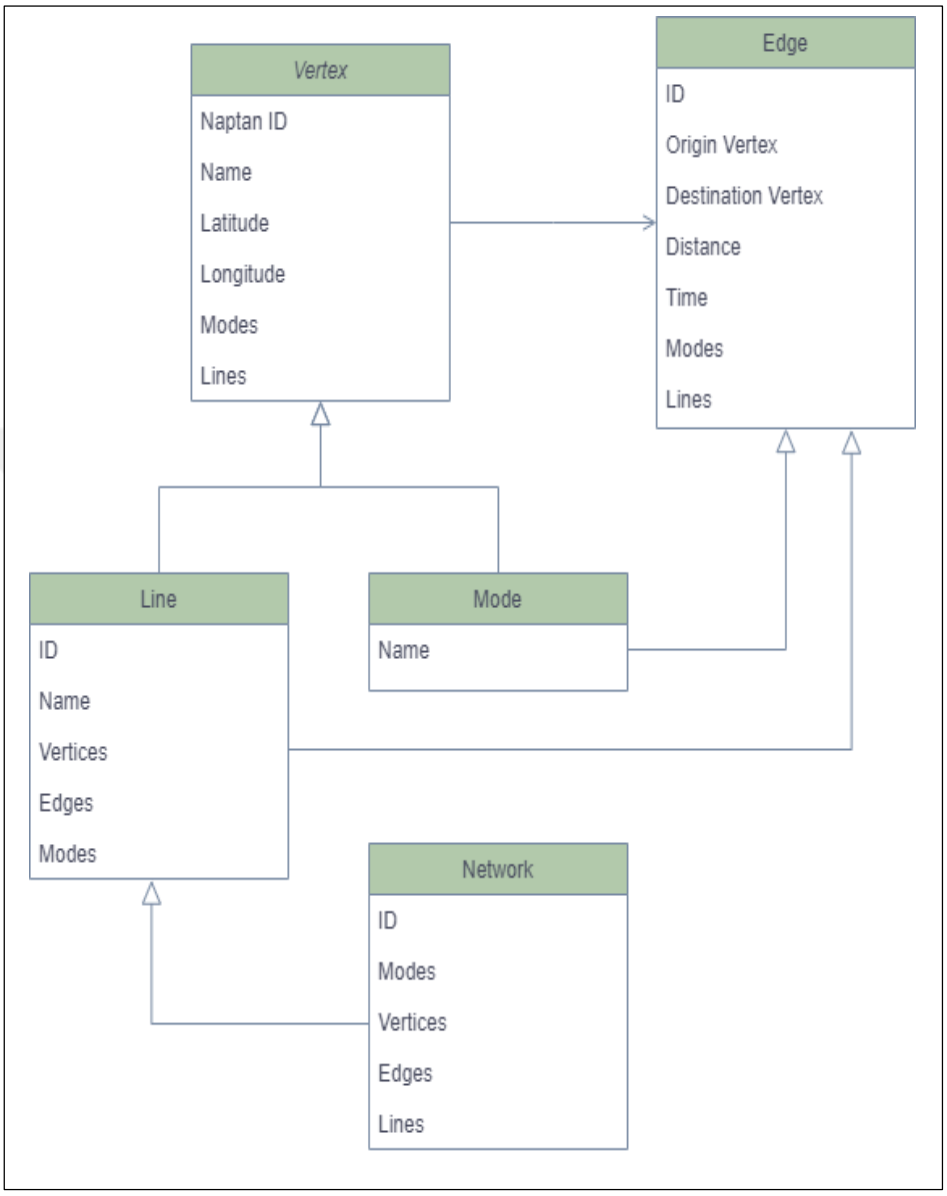


Figure 5.5. Class data structure of Greater London's TFL network.

Algorithm 10: Extract and create Greater London network

```
Inputs:
Output: network.
List_of_lines ← requestOne()
for each line in List_of_lines do
    line_response ← requestTwo()
    outbound ← getOutbound(line_response)
    inbound ← getInbound(line_response)
    line_stations ← removeRedundancies(outbound, inbound)
    line.mode ← getMode(line_response)
    for each station Line_stations do
        vertex ← createVertex(station)
        vertex.latitude ← getLatitude(station)
        vertex.longitude ← getLongitude(station)
        vertex.mode ← setMode(line.mode)
        line_vertices ← addVertex(vertex)
    end
end
for each line in List_of_lines do
    for i=0, I < |line_vertices| do
        origin ← line_vertices[ i ]
        destination  ← line_vertices[ i+1 ]
        journey_response ← requestThree(origin, destination)
        edge ← createEdge(origin, destination)
        edge.time ← setTime(journey_response)
        edge.distance ← haversine(origin, destination)
        edge.mode ← setMode(line.mode)
        line.edges ← addEdge(edge)
    end
end
network ← createNetwork(List_of_lines)
end
```

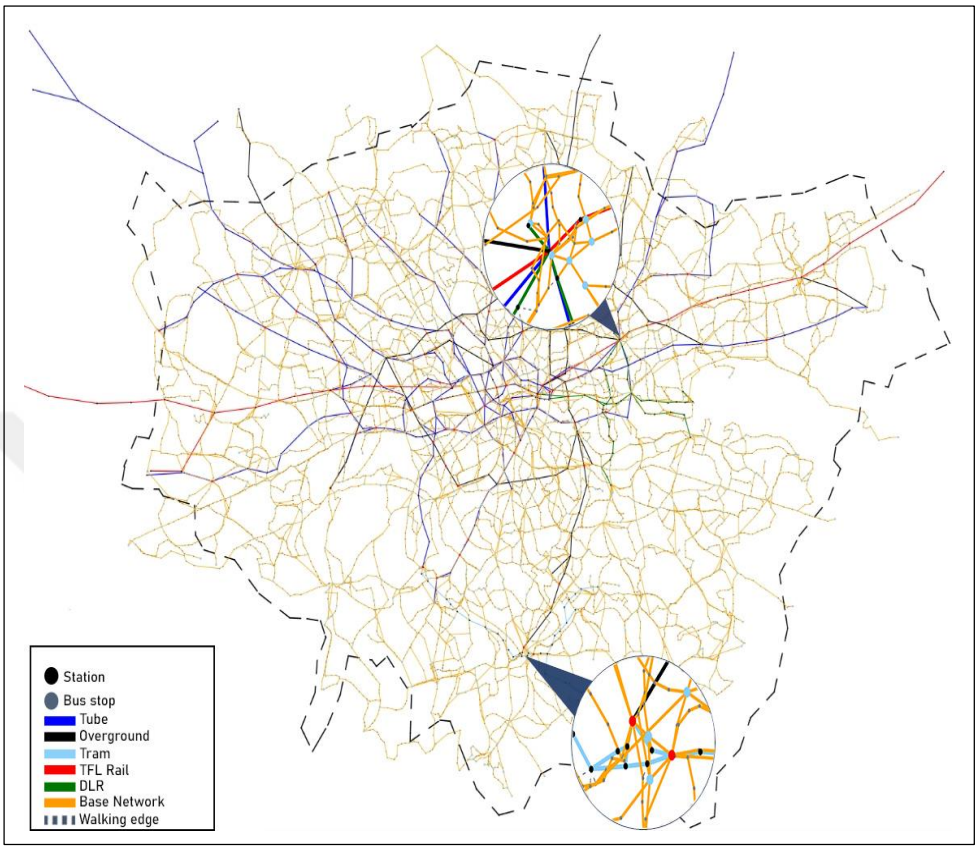
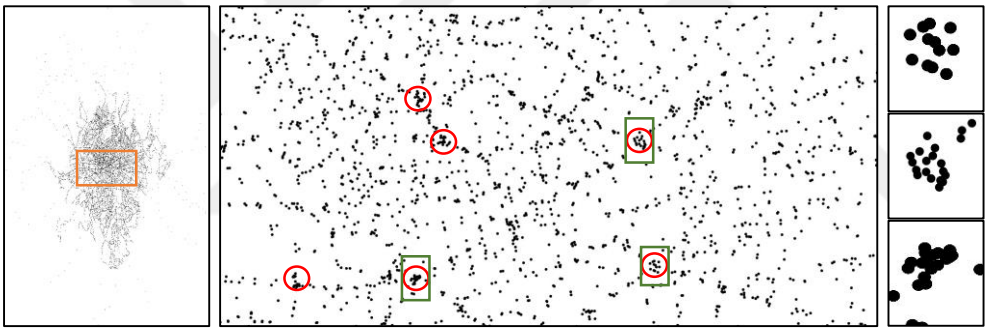


Figure 5.6. Grater London's MPTN network.

6 METHODOLOGY

6.1 Phase One: Clustering

In an MPTN dataset, each transportation network is represented by a set of lines and each line is a sequence of stops or stations. Clusters are formed in areas where several stops from different transportation lines are located close to each other. Figure 6.1 shows an example of clusters on the MPTN of Paris. Figure 6.1. Figurea shows a general perspective for the whole network. Figure 6.1. Figureb is the zoom of the area inside the orange rectangle in Figure 6.1. Figurea where some of the clusters are indicated by the red circles. Figure 6.1. Figurec shows high zooms of three clusters indicated by the green rectangles and red circles.



(a) Paris dataset (b) Zoom inside orange rectangle (c) High zoom

Figure 6.1. Figure. Clusters on MPTN of Paris.

A property of MPTN datasets is that the distances between successive stops on a line are nearly equal. In a bus network, for example, these distances are about 500 meters. Other modes of transport have different distance values, resulting in an invariant node density in the dataset. The aforementioned algorithms (*acdt*, *triclust* and *autoclust*) perceive these datasets as one or more large clusters and fail to find feasible clusters for MPTNs. However, the objective is to identify clusters, as shown in Figure 6.1. Figurec. Thus, there is a genuine need for an algorithm that does not depend entirely on the density of nodes. To identify these clusters, the proposed

algorithm uses the Delaunay triangulation method, which has been shown to be an effective tool to detect the proximity between data points (Deng et al., 2011; Estivill-Castro & Lee, 2000; D. Liu et al., 2008). The details of using Delanay triangulation are explained in the next section.

6.1.1 Spatial Clustering for Public Transit

The proposed algorithm is called Spatial Clustering for Public Transit (*scpt*). In this section, first the notation is given, then the algorithm is explained using the definitions, and at the end of the section a complexity analysis is provided.

Assume that an MPTN consists of the following modes of transport: public bus, tramway, rapid bus and subway. The stops and stations of the networks are represented by spatial data points in a 2D Euclidean space. The union of these points represents the whole MPTN dataset. Let $S = \{p_1, p_2 \ldots p_N\}$ be the set consisting of N spatial points, $DT(S)$ be the Delaunay triangulation of S, and $E = \{e_1, e_2 \ldots e_M\}$ be the set of Delaunay edges. Each Delaunay vertex corresponds to a single point p_i in S. Delaunay edges do not correspond to the actual roads or links of the MPTN.

Definition 1. *Initial_Mean(DT)* denotes the mean for the lengths of all edges in *DT(S)* and is given as

$$Initial\_Mean(DT) = \frac{\sum_{i=1}^{N} l_i}{N} \quad (1)$$

where *li* is the Euclidean length of e_i

Definition 2. *Initial_Std_Dev(DT)* denotes the standard deviation for the lengths of all edges in *DT(S)* and is given as

$$Initial\_Std\_Dev(DT) = \sqrt{\frac{\sum_{i=1}^{N} (l_i - Initial\_Mean(DT)^2)}{N}} \quad (2)$$

Definition 3. *Upper_Threshold* denotes the first threshold value and given as

$$Upper\_Threshold = Initial\_Mean(DT) + Initial\_Std\_Dev(DT) \quad (3)$$

if an edge length *li* is greater than *Upper_Threshold* the edge is removed. This operation discards very long edges located on the border and on the gaps of the dataset. Those long edges create an incoherence in statistical features extracted from the dataset.

Definition 4. Mean(DT) and Std_Dev(DT) denote the mean and the standard deviation of the length of all edges left in DT(S) after the removal operation.

Definition 5. Lower_Threshold denotes the second decision value used to decide on whether an edge should be removed or not and given as

$$Lower\_Threshold = Mean(DT) - (\alpha \cdot Std\_Dev(DT)) \quad (4)$$

where α is the scaling factor of *Std_Dev_(DT)*. The sizes of the clusters are directly affected by α. When α is small *Lower_Threshold* gets closer to the mean and more edges are eliminated. In most of the experiments, α is set to 1/2, the effects of using different α values are explained in Results.

Removing the edges shorter than *Lower_Threshold* from the graph results in scattering of clusters on the graph. Some of the clusters are weakly connected via chains and junction points.

Definition 6. An edge that does not belong to any triangle is called a free edge. Figure 6.2 shows three types of free edges: (a) a disconnected edge (b) a leaf edge (c) a chain edge.

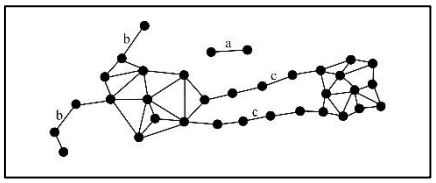


Figure 6.2. Free edges

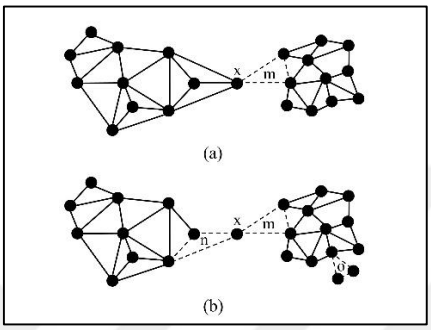


Figure 6.3. Junction points (a) single tri-angle (b) multiple triangle

Definition 7. A vertex v_i is called a junction point if one of the triangles to which it belongs is connected to the other triangles only via v_i. As can be seen in, Figure 6.3 there are two types of junctions: a junction for a single triangle and a junction for multiple triangles. Figure 6.3.a shows the first type. The triangle *m* is connected to the neighboring triangles only via the vertex *x*. The vertex *x* is therefore a junction point. To separate two subgraphs, the edges of *m* that are connected to *x* are removed. Figure 6.3.b shows an example of the second type of junction point. In this case, both triangles m and n satisfy the condition given in Definition 6. To separate the subgraphs, the Euclidean distances between the junction point x and the centers of the triangles m and n are compared and the furthest triangle's edges are removed. There is an exception in the lower right corner of Figure 6.3.b. There is a junction point connected to triangle o, since o is not connected to any other triangle, it is not removed.

6.1.2 The Complexity of *scpt* Algorithm

scpt algorithm consists of five main steps. The complexity of each step is explained below:

Step1: Initialization

- Perform a Delaunay triangulation on the set *S* and obtain *DT(S)*. This takes *O(N log N)* operations. As already mentioned, *N* is the number of spatial points of the MPTN dataset.
- Calculate the initial mean of the lengths of all edges, *Initial_Mean(DT)* (Definition 1). The complexity of this computation depends on the number of Delaunay edges (*M*). The number of Delaunay edges is linearly related to *N*. Consequently, calculating the initial mean requires *O(N)* operations.
- Calculate the initial standard deviation, *Initial_Std_Dev(DT)* (Definition 2). Similar to the initial mean calculation, this takes O*(N)* operations.
- Check each edge and remove it if it is longer than *Upper_Threshold* (Definition 3). As with previous calculations, the complexity of the removal process depends on the number of Delaunay edges and requires *O(N)* operations.

Step 2: Compute the *Lower_Threshold* (Definition 5).

- Calculate the mean of the lengths of all edges, *Mean(DT)* (Definition 4). Although the number of Delaunay edges decreases after Step 1, the extent of the reduction varies as the network changes. In the worst case (without reductions), the complexity is *O(N)* operations.

- Calculate the standard deviation of all edges *Std_Dev(DT)* (Definition 4). Similar to the previous process, it requires *O(N)* operations.
- Calculate *Lower_Threshold*, this calculation has constant complexity of (*O(1)*).

Step 3: Remove edges

- Check each edge and remove it if it is longer than *Lower_Threshold*. This needs *O(N)* time.

Step 4: Remove free edges and junction points

- Remove free edges (bridges, leaves etc.) and the edges of single triangles (Definition 6). In the worst case, this process takes *O(N)* time.
- Remove edges at junction points (Definition 7). In the worst case, this process needs *O(N)* time.

Step 5: Clustering

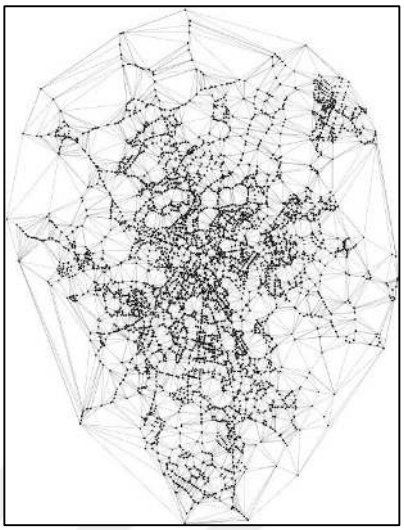
- Execute BFS algorithm and create a cluster for each connected subgraph. The BFS algorithm requires *O(M+N)* operations if an adjacency list is used in the implementation, where M is the number of Delaunay edges and *N* is the number of vertices. Since the number of Delaunay edges depends on *N*, the total complexity of this process is *O(N)*.

After this analysis, it can be concluded that the time complexity of the *scpt* algorithm is determined by the complexity of the Delaunay triangulation, which requires *O(N log N)* operations. The pseudocode of *scpt* is shown in Algorithm 11.

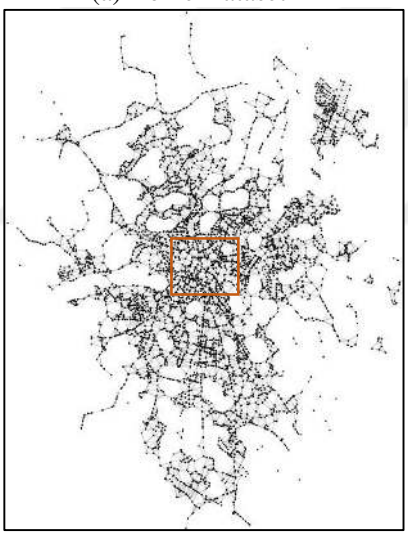
Algorithm 11: scpt clustering.

```
Input: Set of 2D spatial points S = {v1, v2, v3…, vn}
Output: Set of clusters C = {c1, c2, c3…, cm}
DT ← DelaunayTriangulation(S);
Calculate Initial_Mean(DT);
Calculate Initial_Std_Dev(DT);
Upper_Threshold ← Initial_Mean(DT) + Initial_Std_Dev(DT);
RemoveEdges(Upper_Threshold);
Calculate Mean(DT);
Calculate Std_Dev(DT);
Lower_Threshold ← Mean(DT) - (α · Std_Dev(DT));
RemoveEdges(Lower_Threshold);
FindAndRemoveFreeEdges();
FindAndRemoveChains();
FindAndRemoveSingleTriangles();
iterator, i ← 0;
while S size >  do
    vertex v ← S (iterator);
    If the size of neighbors of v > 0 then
        BFS(v);
        ci ← BFS traversed vertices;
        remove vertices of ci from S;
        C ← ci;
        i++;
    else
        add v to noise list;
    end
        iterator++;
end
```

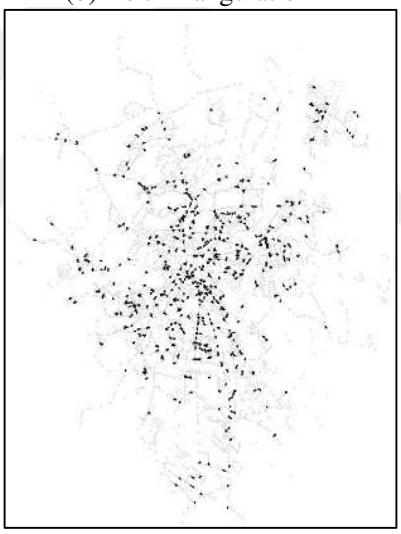
(a) Rome Dataset



(b) Del. Triangulation

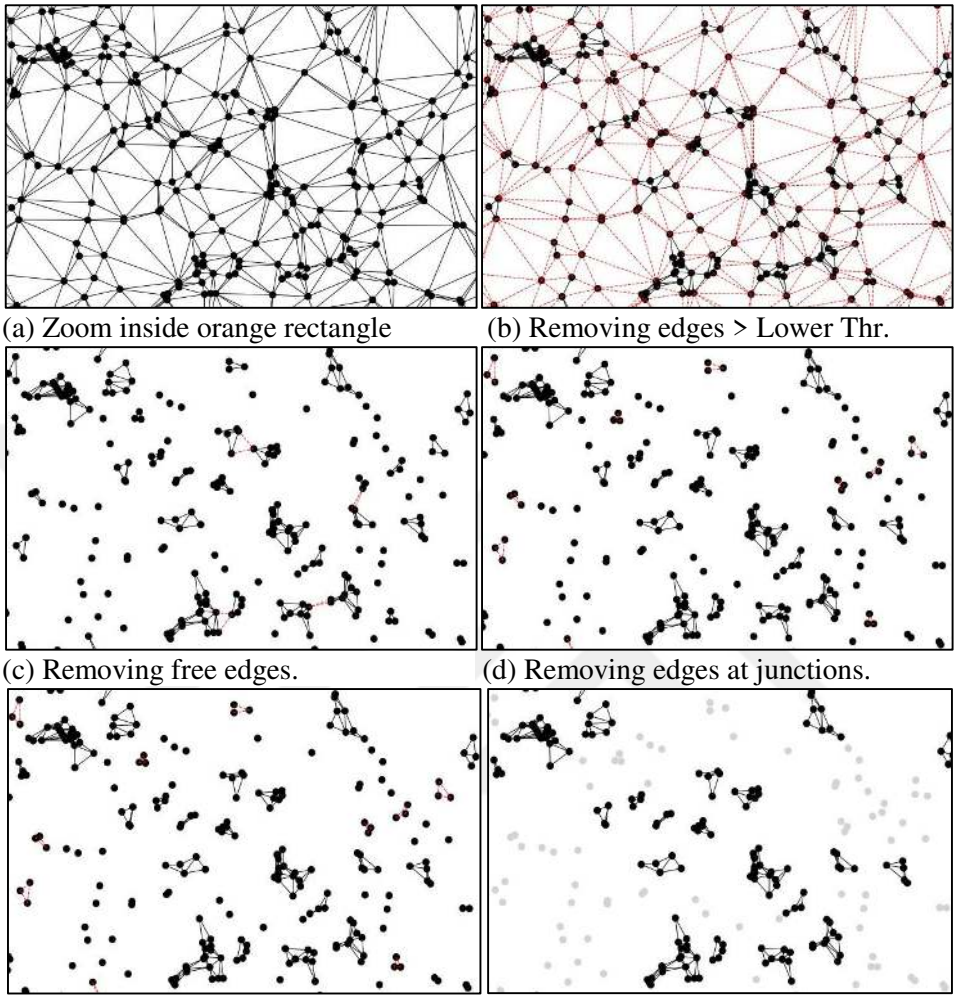


(c) Removing edges > Upper Thr.



(d) Clusters

Figure 6.4. scpt steps on Rome's Dataset.



(a) Zoom inside orange rectangle (b) Removing edges > Lower Thr.

(c) Removing free edges. (d) Removing edges at junctions.

(e) Removing edges of single triangles. (f) Clustering

Figure 6.5. Zoom of scpt steps inside orange rectangular on Figure 20.c.

Figure 6.4 shows the visualization of each *scpt* step on the Rome dataset. The Delaunay edges at the border of the MPTN dataset are noticeably long. These edges are the unintended consequence of triangulation. As shown in Figure 6.4.c, the first removal operation gets rid of these extremely long edges. The clustering step removes more edges and identifies the clusters in the dataset. Figure 6.5 shows the zooms into the parts indicated by the orange rectangle in Figure 6.4.c. Each frame in

the figure shows the transformation of the MPTN dataset by the processes described. In all frames, the edges to be removed are indicated by red dashed lines.

6.2 Phase two: Generating a hub and spoke network

In this Thesis, the hub-level network is based on fixed public transport networks FPTN (rail-based transport networks) and public bus networks PBN. Initially, priority is given to the FPTN to locate the hubs and hub lines. Then, in the areas where there are no FPTN lines, PBN-based hubs and hub lines are generated. The spoke-level network is created using the single allocation method; the spokes are assigned to the nearest hub.

The clusters generated by *scpt* are used to create PBN hub and spoke lines. These lines collect flow from the centroids of the clusters and direct it into the hub-level network. The station closest to the center of the cluster is considered as the centroid.

6.2.1 Hub-level network

The method of creating the hub-level network is inspired by the growing network model. Growing networks have a root vertex that acts as the source of distributed flow. The network grows outwards from the root vertex, and a new vertex is added to the network based on optimized paths through the network to the root vertex (Gastner & Newman, 2006). This model was used in the work of (Majima et al., 2017). There, metaheuristic methods were used to find a single-hub and use it as a root. Then lines are created to connect the spokes to the hub, using the principles of the growing network.

Since this work focuses on the problem of locating multiple hubs, a root vertex is first found and selected as hub, and the other vertices as candidate hubs. Then paths are generated between the hub and the candidates, the best path is selected as a hub line and its target candidate is selected as a hub. With the newly selected hub as the root, the process is repeated recursively until no hub candidate or

valid path is available. The main inputs used in this algorithm are the FPTN network, the centroids of the clusters generated by *scpt*, the demand matrix and the distance matrix.

6.2.1.1 Locating the root hub

In an FPTN, an interchange station is connected to multiple lines, and in addition, some interchange stations may serve multiple modes. This type of station can consolidate demand from multiple lines and divert it to other lines, making it suitable as a hub. However, two interchange stations may be too close to each other. If these stations are both considered as hubs, it will lead to low economies of scale. It is well known in the HLP literature that the greater the distance between hubs, the higher the economies of scale (Alumur & Kara, 2008). On the other hand, a station that serves only one line may be located in an area with high demand density and has the potential to be a hub as well. In order to classify the stations according to their potential as hubs, this work uses the weighted sum model (WSM) to evaluate the stations according to four criteria:
Connectivity (*C*), Multimodality (*M*), Demand Coverage (*DC*), and Distance (*D*).

Let $F = \{f_1, f_2 \ldots f_N\}$ be a set of *N* vertices, where each vertex f_i in *N* represents a station in the FPTN. The performance of each vertex, sf_i, is calculated according to Equation 5:

$$sf_i = w_0\left(\frac{C_i}{C^{max}}\right) + w_1\left(\frac{M_i}{M^{max}}\right) + w_2\left(\frac{DC_i}{DC^{max}}\right) + w_3\left(\frac{D^{min}}{D_i}\right) \quad (5)$$

where C_i is the number of lines connected to f_i, M_i is the number of modes served by f_i, DC_i is the sum of demand originating from each vertex located within the catchment area of f_i, which has a radius of 7 km, and D_i is the sum of the actual road distance from each vertex located within the catchment area of f_i. $w_{0\text{-}3}$ are the weights applied to each criterion. For simplicity, $w_{0\text{-}3}$ are set to 1/4. Here, linear normalization

is applied to each criterion, as this is one of the best options for use with *WSM* according to (Vafaei et al., 2018). The vertex with the highest performance score is considered as the root hub and included in the hubs list (HL). The remaining vertices are sorted in descending order of performance score and included in the candidate hubs list (CHL).

Short hub lines reduce the benefit from economies of scale, so a uniform distribution of hubs in the network is required. The literature recommends that a radial catchment area of 6.4-8 km is appropriate for a hub (J. Yu et al., 2011) Therefore, for the test conducted in this Thesis, the radius of the catchment area of a vertex is set to 7 km for simplicity.

6.2.1.2 Locating hubs and hub lines

In this part, a recursive algorithm that finds a hub line is presented. First, the candidate hubs that are in the catchment area of the root hub are removed from the CHL. Second, for each candidate hub, a path between the root hub and the candidate hub is created based on Dijkstra's Shortest Path (DSP) algorithm. If the resulting path is multimodal, the edge connecting the modes is temporarily removed from the graph and the process is repeated until a single-mode path is found or no path exists. The reason that only single-mode paths are accepted is to ensure that demand is carried continuously along the hub line without transfers. All accepted paths are added to the list of valid paths (VPL) and evaluated based on three criteria: Travel Time (*TT*), Path's Demand Coverage (*DP*), and Zones Served (*ZS*).

Let P = $\{p_1, p_2 \ldots p_M\}$ be the set of M paths included in the VPL. The performance of each path sp_i is calculated using Equation 6:

$$sp_i = w_0 \left(\frac{TT^{min}}{TT_i}\right) + w_1 \left(\frac{DP_i}{DP^{max}}\right) + w_2 \left(\frac{ZS_i}{ZS^{max}}\right) \quad (6)$$

where TT_i is the travelling time on edge p_i. DP_i is the sum the demand originating from each vertex $\in p_i$. Since the origin of the path is the root hub and the destination is a candidate hub, the demand originating from the catchment area is calculated. The nodes that are both on the path and in the catchment area are considered only once to avoid double coverage of demand. ZS_i is the number of zones served by the path, if a zone is traversed by the path, it is considered served. $w_{0\text{-}2}$ are the weights applied to each criteria, and are set to 1/3. The path with the highest performance score is selected as the hub line. The candidate of the selected hub line is considered a hub and added to HL. The process is repeated recursively until no candidate remains or no valid path exists. The pseudo code of the algorithm is shown in Algorithm 12. Some lines found in the first iterations of the algorithm may contain vertices in the middle, which are later called hubs. Therefore, updates are required to determine the final state of the hub lines.

It is almost impossible for the network to be fully connected at the hub-level in the MPTN, as is the case in other areas such as air transport. To maximize the connectivity of the network at the hub level, previously accepted paths in VPL connecting two hubs are also considered as hub lines.

Algorithm 12: Locating FPTN hub lines and hubs.

Input: HL the list of hubs. CHL the list of candidate hubs.
DM the demand matrix. ND the network data.

Output: FHLL the list of FPTN hub lines, and HL

```
Function getHubLine(HL, CHL, DM, ND)
    RootHub ← HL[n];
    VPL ← ∅;
    removeCandidates(RootHub, CHL, ND);
    foreach CH ∈ CHL do
        path ← DSP (RootHub, CH);
        if path mode ≠ 1 then
            path ← singleModePath(RootHub, CH, ND);
        end
        if path mode ≠ ∅ then
            append(VPL, path);
        end
    end
    bestPath ← weightedSum(VPL);
    append(FHLL, bestPath);
    append(HL, bestPath.destination);
    if CHL ≠ ∅ then
        getHubLine(HL, CHL, DM, ND);
    end
end Function
```

6.2.1.3 Bus Mode-based Hub Network

To create PBN-based hub network, the first step is to determine the zones that are eligible to locate a bus hub. If a zone does not have an FPTN station and the center of the zone is not within the catchment area of an FPTN hub, then the zone is eligible. Locating hubs in these zones is straightforward: the centroid that is closer to the center of the zone is considered a hub and included in the HL. Due to the shortcomings of PBN, such as congestion and accidents, it is better to base the hub line on the shortest path. However, this rather serves the interests of passengers. To also consider the interests of operators, the K-Shortest Path Algorithm (KSP) is used.

The paths generated by KSP are evaluated based on demand coverage (*BD*) (in the interest of the operator) and travel time (*BT*) (in the interest of passengers). Let BP = {bp_1, bp_2…bp_{10}} be the set of bus paths generated by KSP for a given origin-destination pair (OD). The performance of each path sbp_i is calculated using Equation 7:

$$sbp_i = w_0 \left(\frac{BT^{min}}{BT_i}\right) + w_1 \left(\frac{BD_i}{BD^{max}}\right) \quad (7)$$

where BT_i is the travel time along the path bpi, BD_i is the sum of demand originating from all vertices on the path bpi. $w_{0\text{-}1}$ are the weights applied to each criteria, and are set to 1/2. For a given origin-destination hub pair (OD), the distance can be very large due to the homogeneous distribution of hubs in the network. For this reason, only ODs with paths shorter than the maximum path length (*maxPL*) are accepted for generating a set of paths with KSP.

For large networks, the computation time of KSP can be very long. To overcome this problem, a validity check procedure is implemented. First, the shortest path is generated using the DSP algorithm. If the length of the shortest path is less than or equal to maxPL, OD is valid. Otherwise, OD is ignored and no hub line is generated. To reduce the search space for KSP, the zones where the shortest path is located are extracted, and only the subgraph located in these zones is considered for KSP. The pseudo code of the algorithm is shown in Algorithm 13.

Algorithm 13: Locating bus hub lines.

```
Input: HL the list of hubs. DM the demand matrix.
       ND the network data.
Output: BHLL the list of PBN hub lines
Function getBusHubLine(HL, DM, ND)
    busHub ← getBusHub(HL);
    BHLL ← ∅;
    foreach H ∈ HL do
        path ← DSP (busHub, H);
        if path lenght ≤ maxLenght then
            path ← singleModePath(RootHub, CH, ND);
            subGraph ← extractSubGraph(path, ND);
            busPathsList ← KSP(subGraph, busHub, H, 10);
            bestBusPath ← weightedSum(busPathsList);
            append(BHLL, bestBusPath);
        end
    end
end Function
```

6.2.1.4 Spoke-level network

After the hub-level network is created, the remaining centroids are allocated to hubs using the single allocation method, where each spoke is allocated to the nearest hub. Two types of spoke lines are created here: FPTN-based spoke lines (FSL) and PBN-based spoke lines (BSL).

Since FPTN does not provide as many options, FSLs are created by using DSP algorithm to generate a path between the spoke and the hub.

In the case of BSL, for a given hub, a path is generated from each spoke to the hub using DSP. The resulting set of paths is filtered to remove redundancies. If a path is completely part of another path, the shorter path is discarded and the path that covers all of them is considered BSLs.

7 RESULTS

7.1 Scpt Experiments and Results

Previous research that use clustering for MPTN design have not used spatial clustering algorithms except the one in (Huang et al., 2018). Though CFS clustering algorithm is used in that study, the details of the clustering results are not reported since the focus of the work was optimization. Because of that, a comparison of scpt results with any previous work that use clustering on MPTN wasn't possible. On the other hand, since *scpt* is a novel spatial clustering algorithm specifically designed for MPTNs, it is compared with three well-known spatial clustering algorithms, *dbscan*, *optics* and *autoclust*. Real-world public transit spatial datasets of Athens, Paris, Sydney, Brisbane, Berlin, Bordeaux, Toulouse, Helsinki, Lisbon and Winnipeg cities obtained from (Kujala et al., 2018) are used to compare the results of all algorithms. These cities are chosen for their sizes because the objective was to test the performance of *scpt* on small, medium and large MPTNs. In addition, most of these cities are situated in different countries and even different continents, which allow us to evaluate the outcome of *scpt* on a variety of infrastructure typologies. It is worth mentioning that *scpt* was tested on all the cities provided in (Kujala et al., 2018), and showed similar performance on all of them. For example, one of the cities was Rome and it was used for presenting the steps of *scpt* in Figure 6.4. For brevity only test results of ten cities are presented in this section. These MPTNs are shown in Figure 4.3. All the code needed for experiments are written in java. *dbscan* is imported from org.apache.commons.math3.mlrg.ustering library (Developers, 2020), *optics* is imported from SPMF library (Fournier-Viger et al., 2016) and *autoclust* is obtained from the link provided in (D. Liu et al., 2008).

The experiments are performed on a PC with i5 Intel core 1.70 GHz CPU, 4 GB RAM, and 64-bit Windows 10 operating system.

The performance of each algorithm on ten cities are shown in Table 7.1. In this table, the first column shows the city and the number of vertices in its MPTN

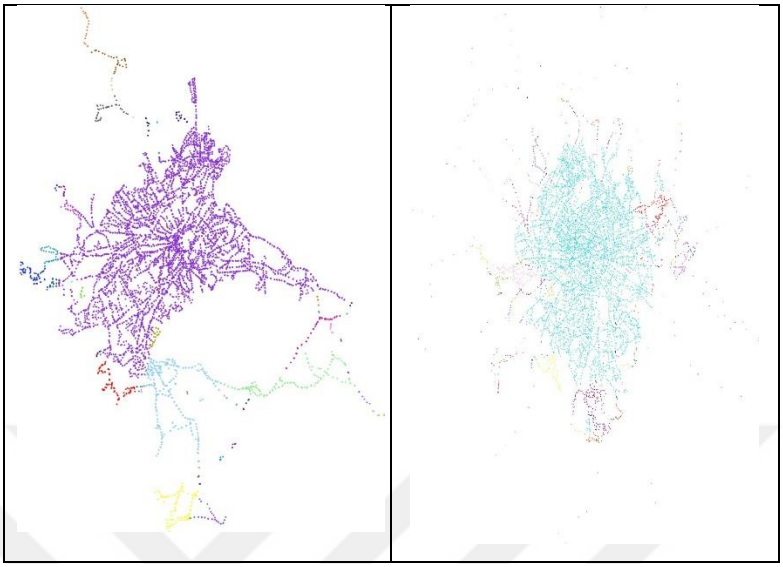
dataset, the second columns shows the number of clusters obtained, and third columns presents the execution time. *scpt* generated the largest number of clusters. The slowest algorithm was *autoclust*. *scpt* is roughly 2 to 3 times slower than *dbscan* and 50 to 100 times slower than optics. However, this performance difference did not cause any problems in practice, since the largest MPTN dataset is processed under 1 minute by *scpt*. Also, note that *scpt* code is unoptimized, while library routines are often well optimized.

Table 7.1. Performance of clustering algorithms

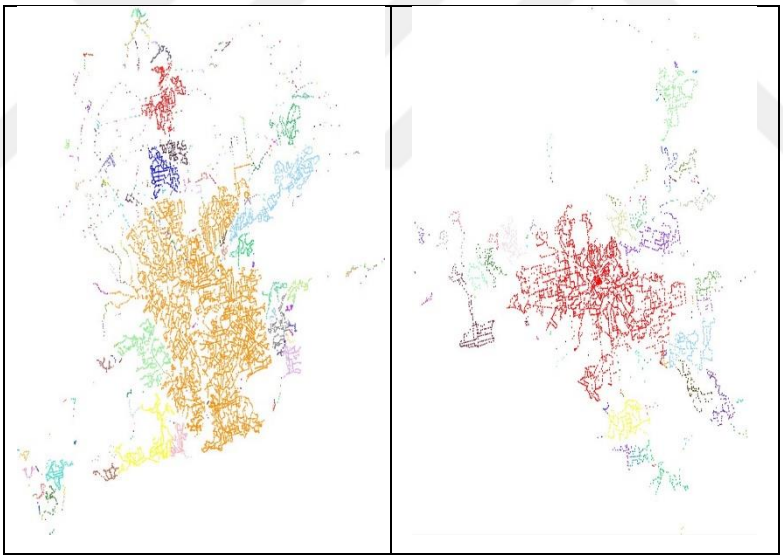
MPTN	Algorithm	Clusters	Time(sec)
	scpt	334	5.1
Athens	autoclust	59	1994.0
6768 vertices	dbscan	180	2.4
	optics	172	0.1
	scpt	595	14.9
Brisbane	autoclust	152	3690.0
9645 vertices	dbscan	246	5.2
	optics	326	0.2
	scpt	757	19.2
Paris	autoclust	273	5565.6
11956 vertices	dbscan	434	5.1
	optics	545	0.3
Sydney	scpt	1481	58.8
24063 vertices	autoclust	417	23506.8
	dbscan	611	25.1
	optics	660	0.6
Berlin	scpt	184	3.0
4601 vertices	autoclust	137	683.7
	dbscan	122	1.3
	optics	107	0.5
Bordeaux	scpt	213	1.4
3435 vertices	autoclust	56	410.2
	dbscan	127	0.4
	optics	157	0.8
Toulouse	scpt	187	4.2
3329 vertices	autoclust	60	423.7
	dbscan	104	0.4
	optics	131	0.2
Helsinki	scpt	403	5.9
6986 vertices	autoclust	197	1593.0
	dbscan	223	2.0
	optics	276	0.1
Lisbon	scpt	414	6.0
7073 vertices	autoclust	121	2036.0
	dbscan	166	2.6
	optics	241	0.3
Winnipeg	scpt	316	3.7
5079 vertices	autoclust	33	966.9
	dbscan	171	0.9
	optics	167	0.2

To make fair comparison, the methods suggested by the authors in their papers are used to compute the required parameters for all algorithms. Specifically, for *dbscan* the *MinPts* parameter was set to 4 in accordance with the original paper (Ester et al., 1996). Also, to set the value for the radius parameter ε the method described in (Ester et al., 1996; Schubert et al., 2017) is used. K-Nearest Neighbor (KNN) algorithm was applied, for each data point the distance was calculated by setting K equal to 4, then, the distances were sorted in descending order and plotted. Slight elbows existed in the plots in the ranges of 50-100, 57-100, 58-70, 70-90, 128-140, 68-86, 72-78, 82-90, 55-60 for Athens, Paris, Sydney, Brisbane, Berlin, Bordeaux, Toulouse, Helsinki, Lisbon and Winnipeg respectively. The parameters are set at the elbows (again following the directions of the researchers) (Schubert et al., 2017). Mean values (Definition 4) computed by *scpt* fell in the same ranges because of that ε was set equal to the Mean values which were 55.81 (Athens), 59.36 (Paris), 59.96 (Sydney), 71.52 (Brisbane), 136.67 (Berlin), 71.49 (Bordeaux), 75.02 (Toulouse), 87.37 (Helsinki), 55.98 (Winnipeg). The same ε and *MinPts* values were used for *optics* runs. Note that these settings improved the results of the compared algorithms.

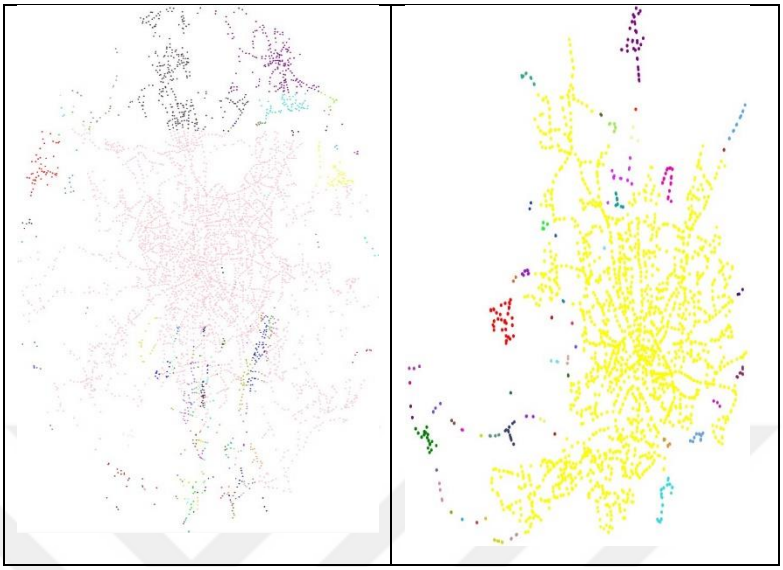
The depiction of all test results for *autoclust*, *dbscan*, *optics*, and *scpt* algorithms were shown in Figures 7.1- 7.4 respectively. Each frame in these figures shows the result of one city, in particular, (a) Athens; (b) Paris; (c) Sydney; (d) Brisbane; (e) Berlin; (f) Bordeaux; (g) Toulouse; (h) Helsinki; (i) Lisbon; and (j) Winnipeg. In these figures, the clusters are designated by different colors, except the *scpt* clusters, which are colored in black and noise in light grey. One can immediately notice that *dbscan*, *optics* and *autoclust* identified many large clusters.



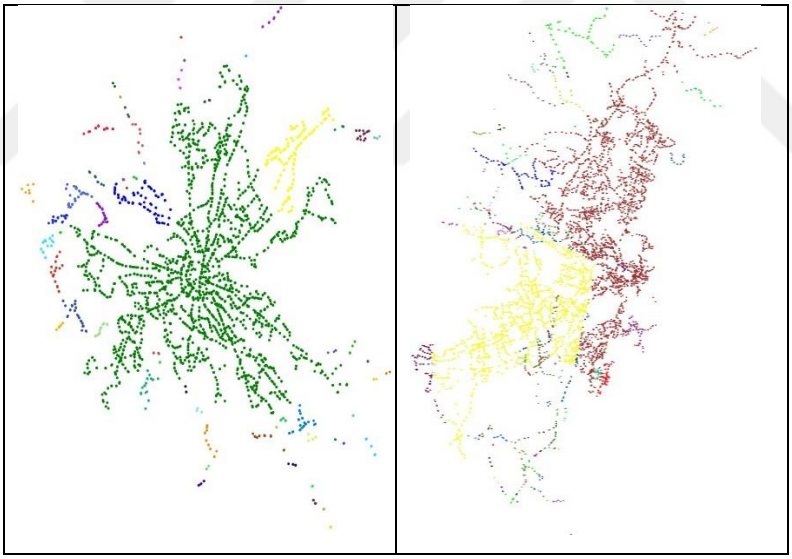
(a) Athens (*autoclust*) (b) Paris (*autoclust*)



(c) Sydney (*autoclust*) (d) Brisbane (*autoclust*)



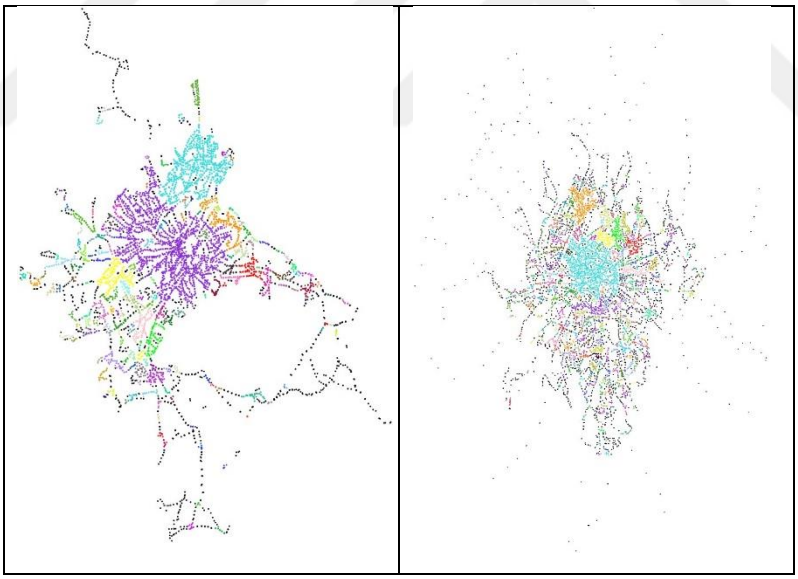
(e) Berlin (*autoclust*) (f) Bordeaux (*autoclust*)



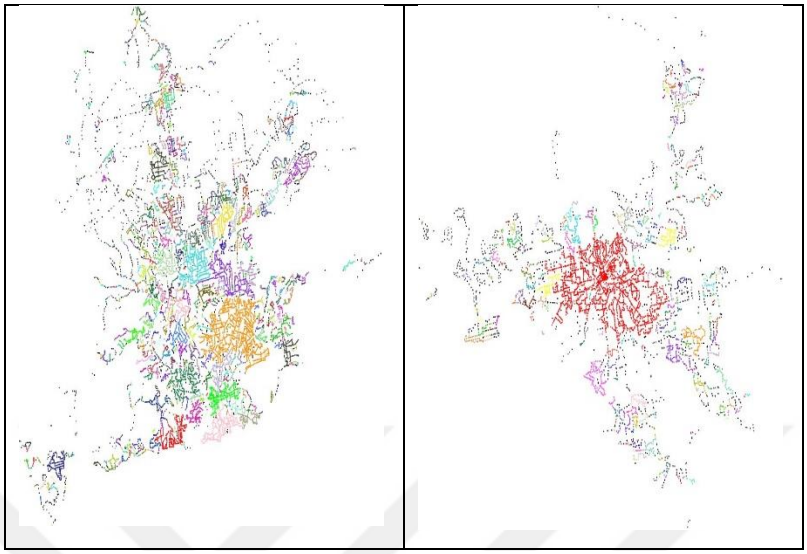
(g) Toulouse (*autoclust*) (h) Helsinki (*autoclust*)

(i) Lisbon (*autoclust*) (j) Winnipeg (*autoclust*)

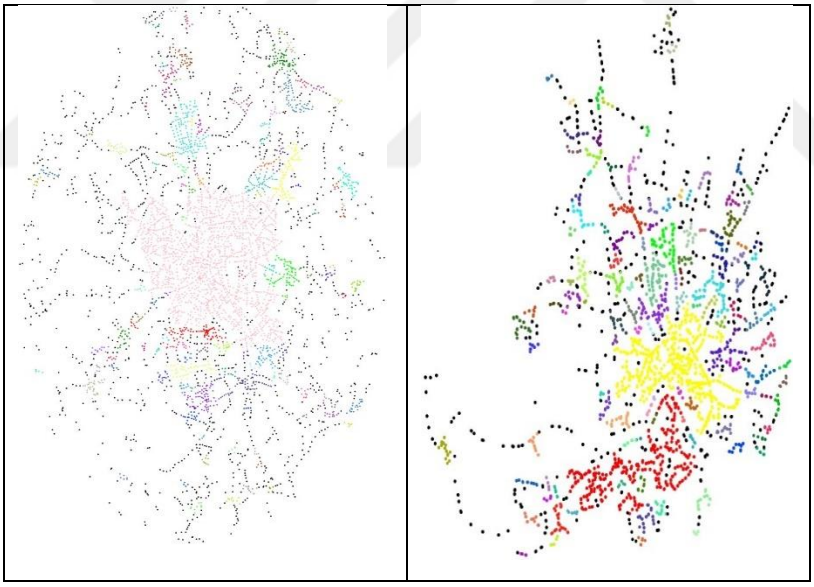
Figure 7.1. Results of autoclust on ten cities.



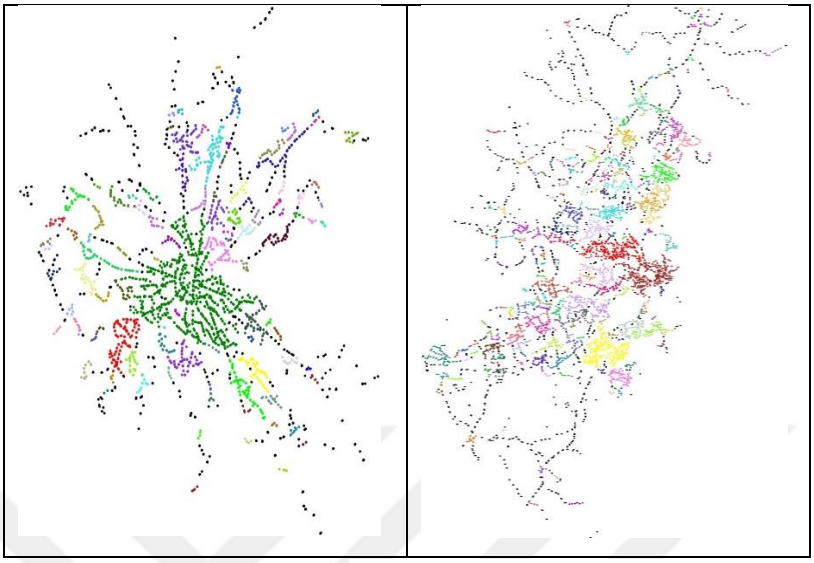
(a) Athens (dbscan) (b) Paris (dbscan)



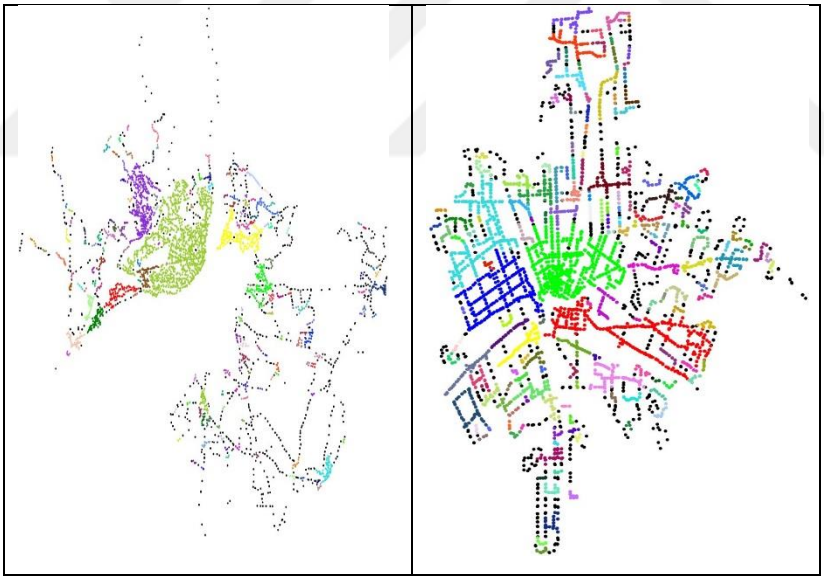
(c) Sydney (*dbscan*) (d) Brisbane (dbscan)



(e) Berlin (*dbscan*) (f) Bordeaux (dbscan)

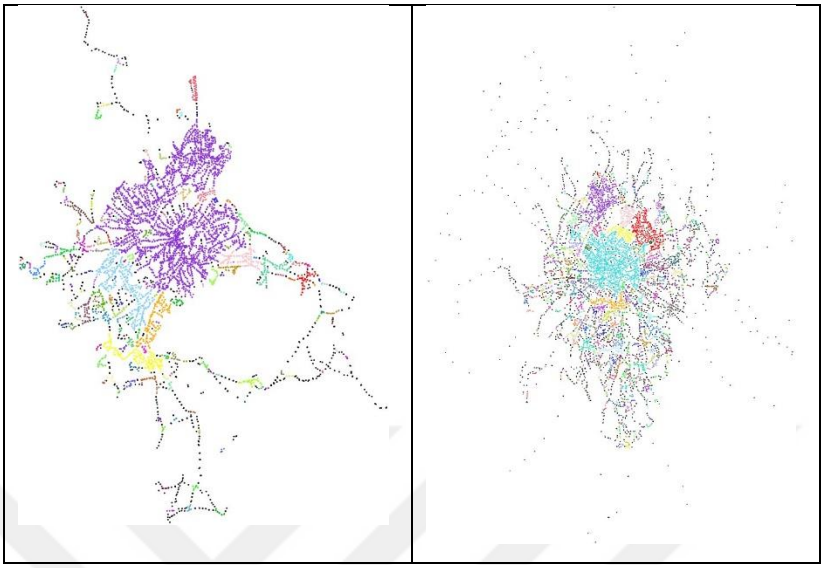


(g) Toulouse (dbscan) (h) Helsinki (*dbscan*)

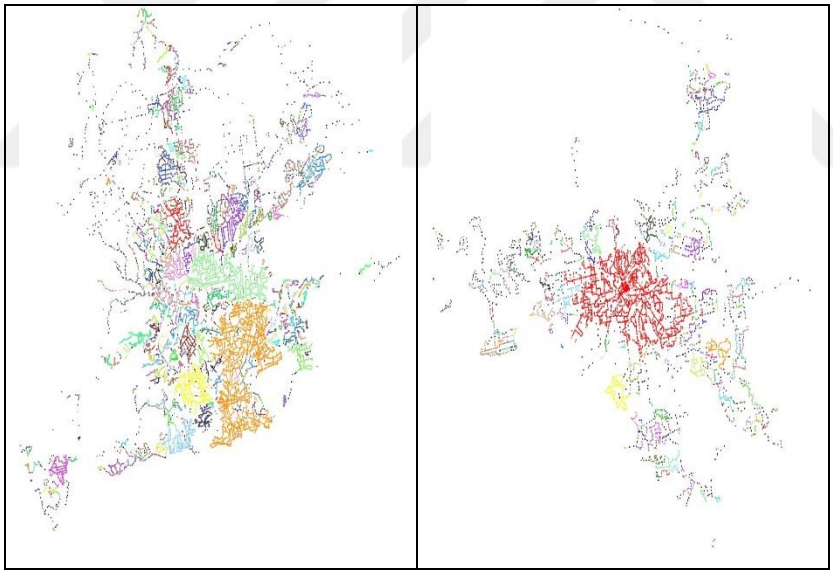


(i) Lisbon (dbscan) (j) Winnipeg (dbscan)

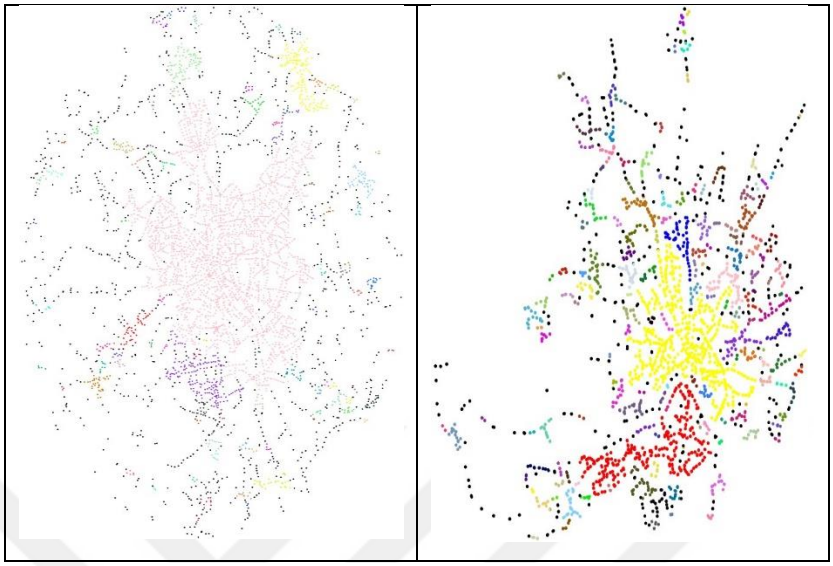
Figure 7.2. Results of dbscan on ten cities.



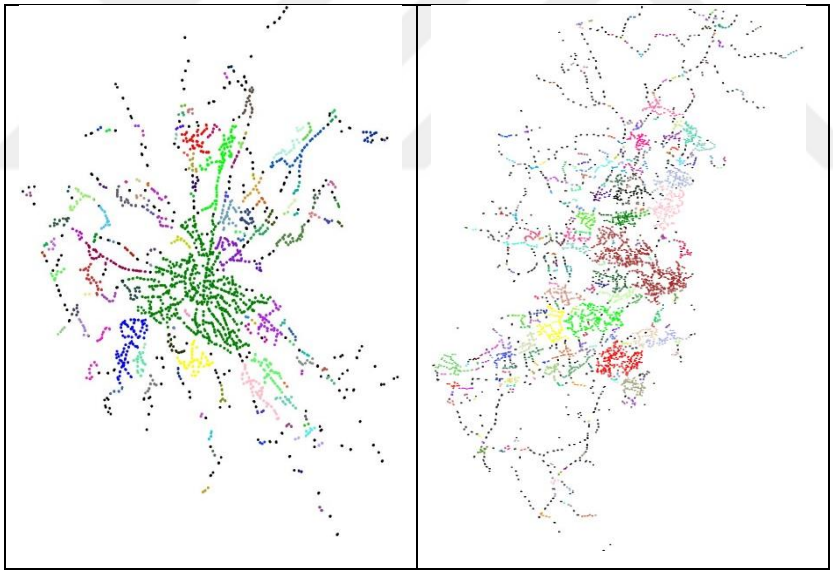
(a) Athens (*optics*) (b) Paris (*optics*)



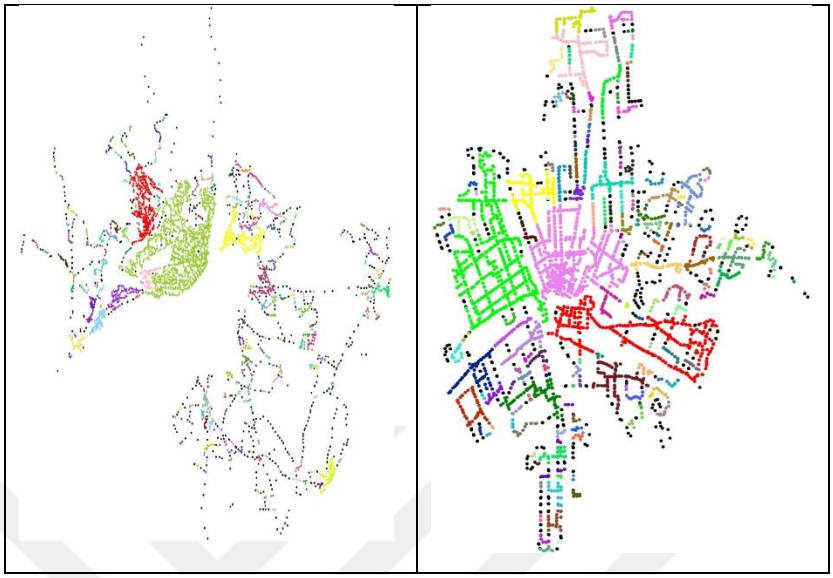
(c) Sydney (*optics*) (d) Brisbane (*optics*)



(e) Berlin (*optics*) (f) Bordeaux (*optics*)

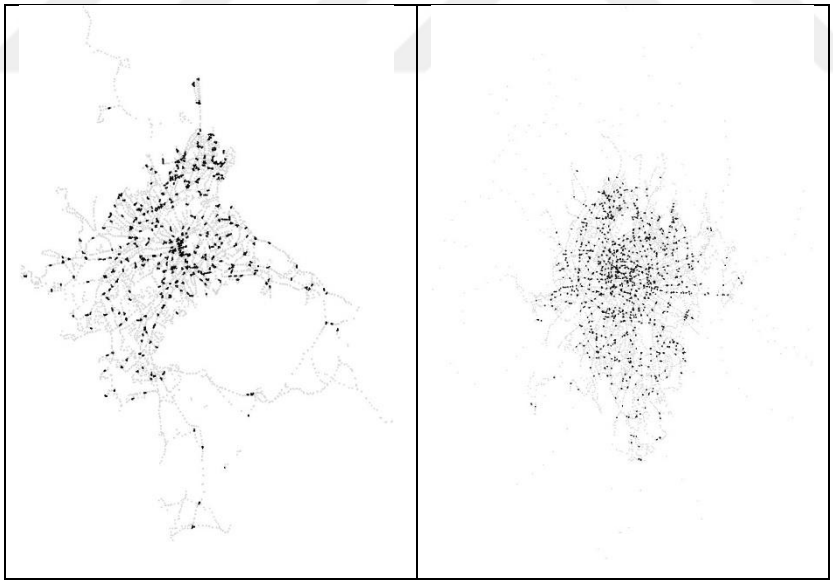


(g) Toulouse (*optics*) (h) Helsinki (*optics*)

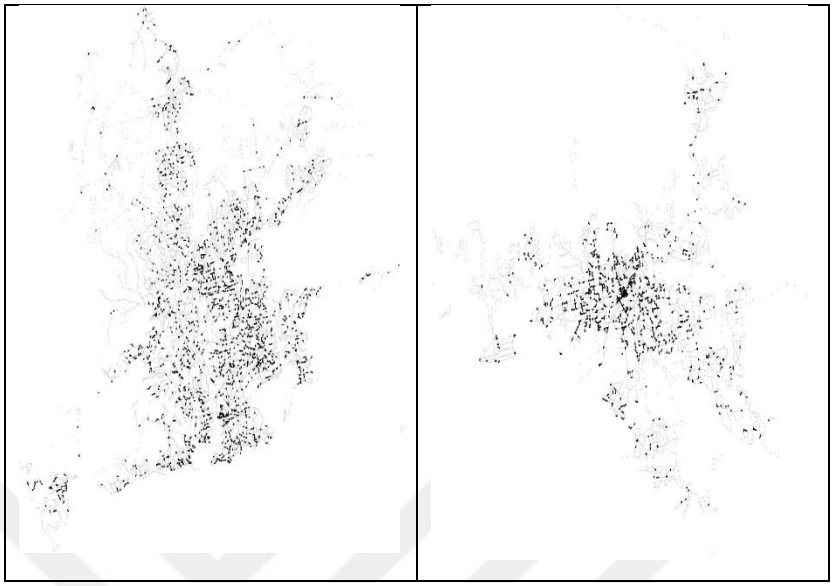


(i) Lisbon (*optics*) (j) Winnipeg (*optics*)

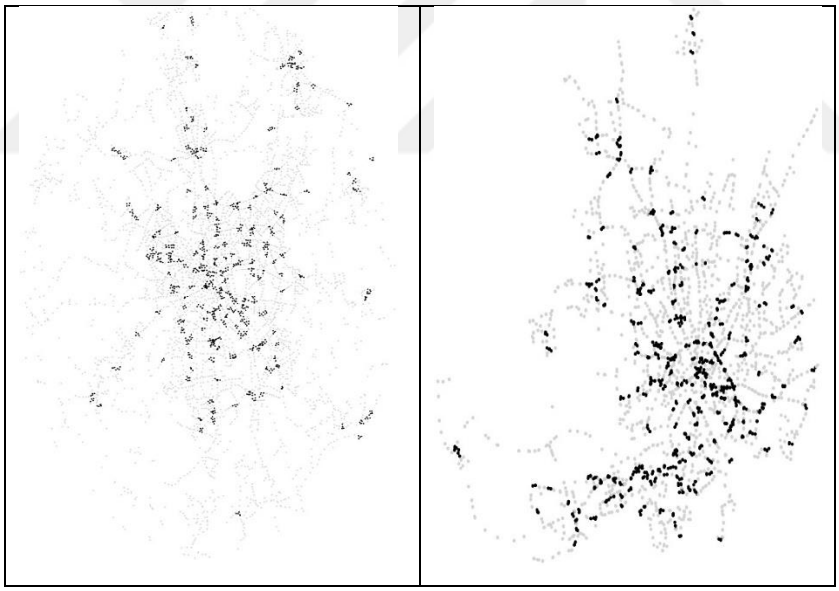
Figure 7.3. Results of *optics* on ten cities.



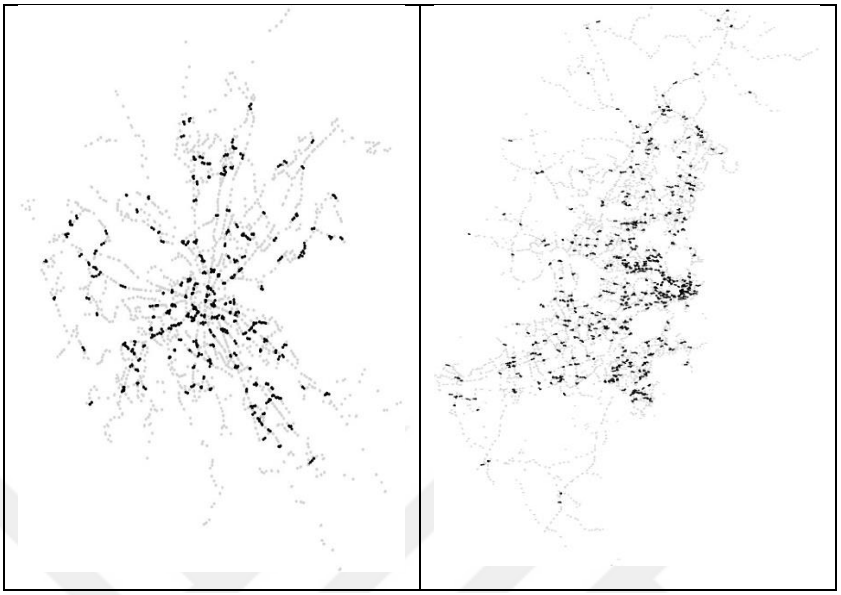
(a) Athens (*scpt*) (b) Paris (*scpt*)



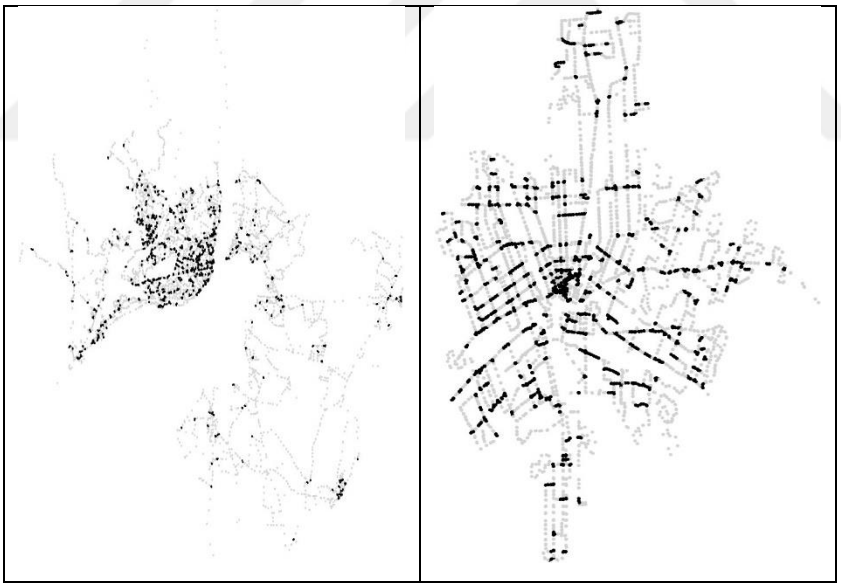
(c) Sydney (*scpt*) (d) Brisbane (*scpt*)



(e) Berlin (*scpt*) (f) Bordeaux (*scpt*)



(g) Toulouse (*scpt*) (h) Helsinki (*scpt*)



(i) Lisbon (*scpt*) (j) Winnipeg (*scpt*)

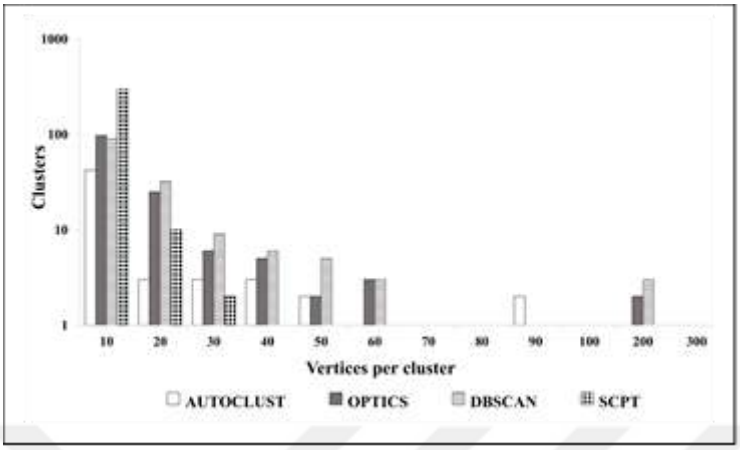
Figure 7.4. Results of *scpt* on ten cities.

Table 7.2. presents the statistics of the experiments for all algorithms, which supports the visuals in Figures 7.1-- Figure 7.4. Table 7.2. is divided into four parts, each part has five rows. The first row shows the algorithm, the second row shows the means of the cluster sizes, the third row shows the standard deviations of the cluster sizes, the fourth row shows the size of the largest cluster, the fifth row shows the percentages of the clusters that consist of 100 or more vertices, the last row shows the percentages of vertices that belong to clusters that consist of 100 or more vertices. The presented data clearly shows that the well-known spatial clustering algorithms generated an unbalanced distribution of cluster sizes in the tested MPTN datasets. For example, for Paris dataset, 1.8% of the clusters generated by *dbscan* contains 37% of the vertices. For the same dataset, 1.4% of the clusters generated by *autoclust* contains 85% of the vertices. In addition to this unbalance in densities, these algorithms tend to generate a few extremely large clusters. For example, the largest cluster for Sydney is found by *autoclust* which consists of 13981 vertices. Locating hub(s) to very large clusters is almost same as locating hub(s) to the initial dataset.

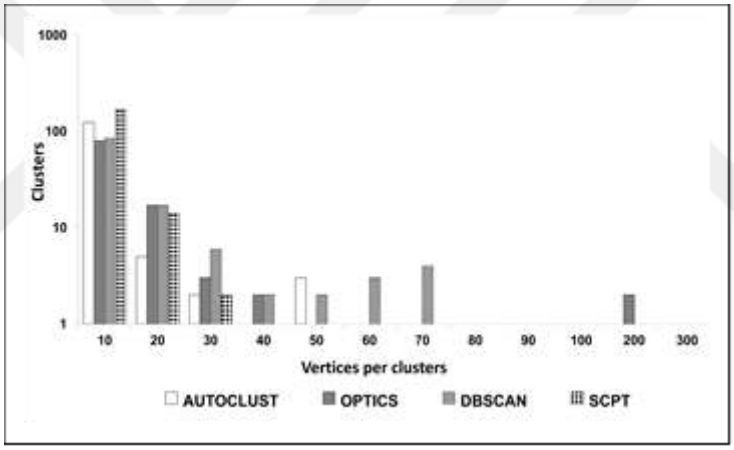
For four cities, histograms for the distribution of cluster sizes are presented in Figure 7.5, each frame demonstrates a histogram of a cities. For brevity, only results of four cities are demonstrated, the choice of the city is based on the size of the dataset, the aim is to show the distribution of the cluster size for small, medium and large MPTN datasets. In these histograms, extremely large clusters (generated by *autoclust*, *optics*, and *dbscan*) are eliminated to provide a clear view of the distributions. All the ranges are divided into consecutive and non-overlapping intervals of 10 and shown on the x axes. y axes represent the count of the clusters that fall into each interval and they are drawn on the log scale. *scpt* generated over 1000 clusters that have 4 to 10 vertices for Sydney and over 100 for the other cities. In fact, all *scpt* clusters are aggregated at most in four ranges. The other algorithms generated cluster sizes in almost all ranges. Table 7.2. and Figure 7.5. shows that the results generated by well-known algorithms are not suitable for locating hubs in MPTN datasets. *scpt* have identified clusters of 100 or less vertices.

Table 7.2. Clustering statistics.

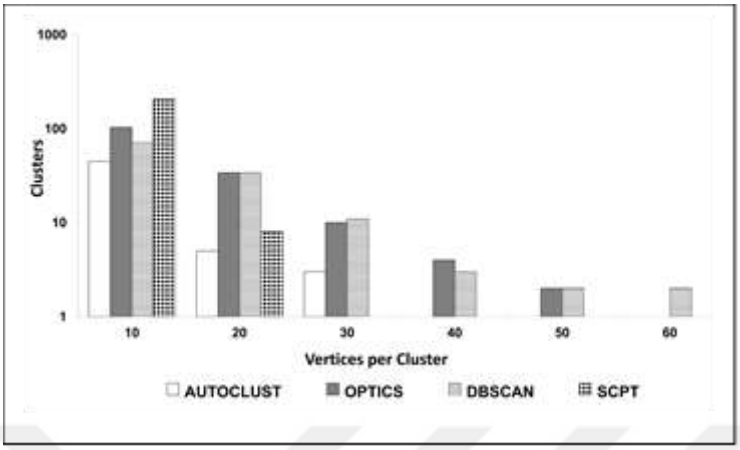
		dbscan	optics	autoclust	scpt
	mean	31.9	34.89	114.63	5.56
	stdev	161.9	249.91	726.09	2.72
Athens	largest	1995	3262	5586	29
	% cl.>100	2.2%	2.9%	5%	-
	% ver.>100	55%	69%	89%	-
	mean	21.06	17.95	43.60	6.06
	stdev	100.40	97	576.81	3.59
Paris	largest	2006	2094	9531	41
	% cl.>100	1.8%	1.2%	1.4%	-
	% ver.>100	37%	40%	85%	-
	mean	34.58	33.09	21.16	5.61
	stdev	128.45	197.29	90.03	2.99
Sydney	largest	2584	4501	13981	60
	% cl.>100	4.9%	3.9%	4.0%	-
	% ver.>100	52%	52%	88%	-
	mean	32.84	26	63.28	5.96
	stdev	240.22	209.22	456.02	6.43
Brisbane	largest	3765	3768	5582	147
	% cl.>100	2.8%	2.1%	7.8%	0.1%
	% ver.>100	57%	54%	86%	4.1%
	mean	27.79	34.11	30.68	6.15
	stdev	148.58	237.04	275.81	3.17
Berlin	largest	1642	2452	3228	24
	% cl.>100	1.6%	2.8%	1.4%	-
	% ver.>100	52%	75%	81%	-
	mean	22.17	18.92	61.25	5.37
	stdev	66.82	72.03	402.56	2.34
Bordeaux	largest	635	803	3019	17
	% cl.>100	1.5%	1.2%	1.7%	-
	% ver.>100	37%	41%	88%	-
	mean	26.85	22.45	55.45	5.47
	stdev	89.14	81.84	337.88	2.56
Toulouse	largest	901	920	2623	25
	% cl.>100	1.9%	2.2%	3.3%	-
	% ver.>100	36%	39%	83%	-
	mean	25.21	21.63	34.95	6.30
	stdev	57.08	74.66	279.54	4.86
Helsinki	largest	592	1086	3424	66
	% cl.>100	5.38%	3.9%	1.5%	-
	% ver.>100	43%	47%	79%	-
	mean	35.03	25.40	58.34	6.07
	stdev	183.26	154.30	375.76	3.40
Lisbon	largest	2266	2291	3968	26
	% cl.>100	4.81%	3.3%	5.7%	-
	% ver.>100	66%	63%	85%	-
	mean	25.71	27.52	153.75	5.78
	stdev	76.91	88.11	772.21	3.21
Winnipeg	largest	684	733	4450	39
	% cl.>100	2.92%	4.1%	8.8%	-
	% ver.>100	41%	53%	93%	-



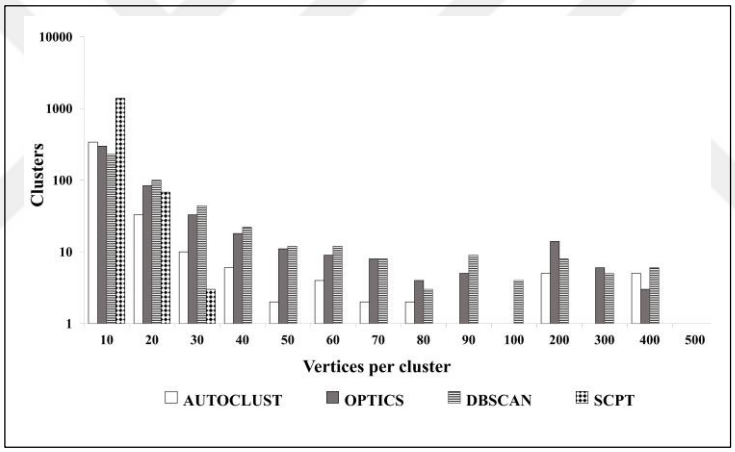
(a) Athens Clusters



(b) Berlin Clusters



(c) Bordeaux Clusters

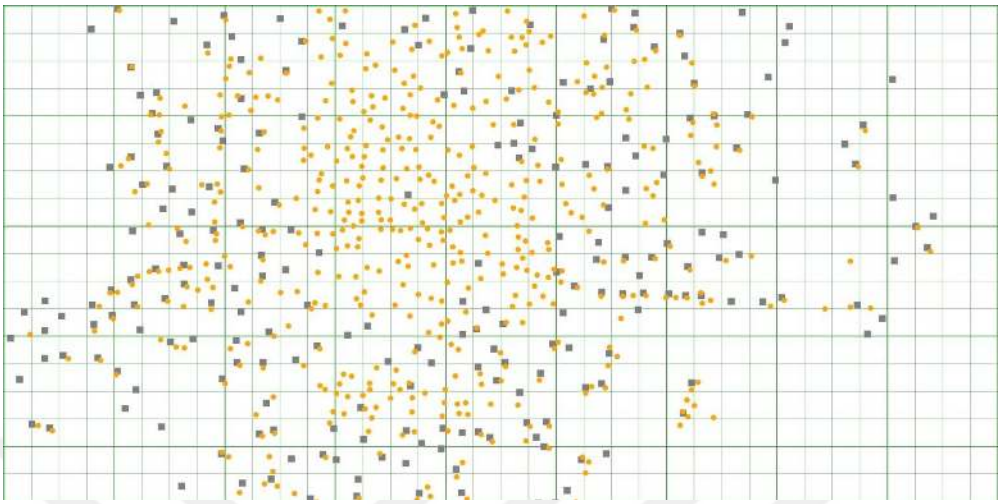


(d) Sydney Clusters

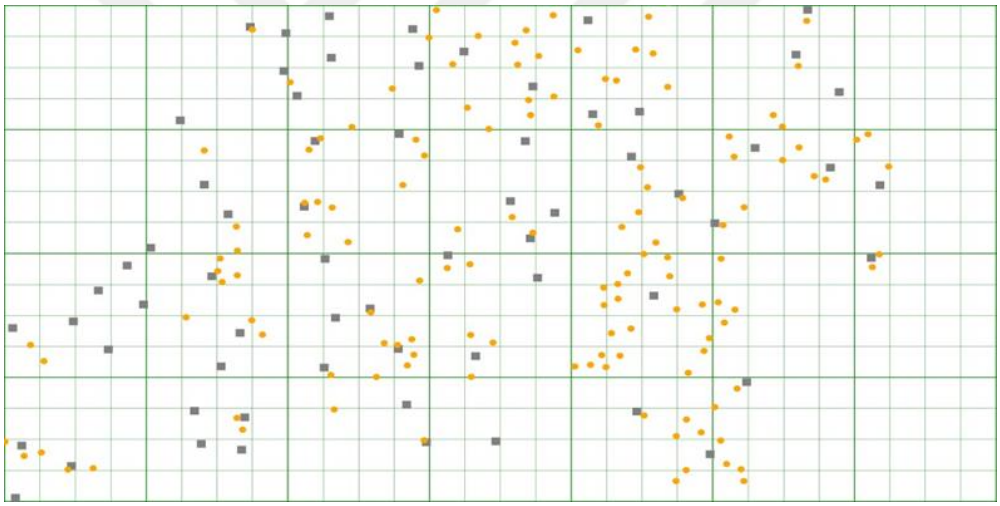
Figure 7.5. The Distribution of Cluster Sizes

Obviously, having small clusters is not enough by itself. Insufficient coverage of geographical area is also undesired. Two grid maps are presented in Figure 7.6. to demonstrate the distribution of the clusters by zooming inside the orange rectangles in Figure 4.2. (Paris) and (Helsinki). To provide a clear view for the reader, only, clusters generated by *dbscan* and *scpt* are shown (*dbscan* results are the most diffused among the others). *dbscan* clusters in Figure 4.2. (Paris) are rare in the center and more concentrated on the edges of the grid, whereas *scpt* clusters

are finely distributed on the center and also located clusters on the borders of the grids close to the clusters of *dbscan*. In contrast, *dbscan* clusters shown in Figure 4.2. (Helsinki), similar to *scpt* clusters are well distributed on the map. However, this distribution is only for the shown part of the map, in the other regions *scpt* clusters are better distributed. Fine grained distribution of clusters on the map increases the alternatives for optimization especially in the regions with high node densities. Also, balanced coverage of the network gives a fair chance of getting a hub in all regions.



(a) Paris centroids



(b) Helsinki centroids

dbscan centroids *scpt* centroids

Figure 7.6. Distribution of the clusters inside the orange rectangles on Paris and Helsinki MPTN datasets (alpha = 1/2). A cluster is represented by its centroid vertex.

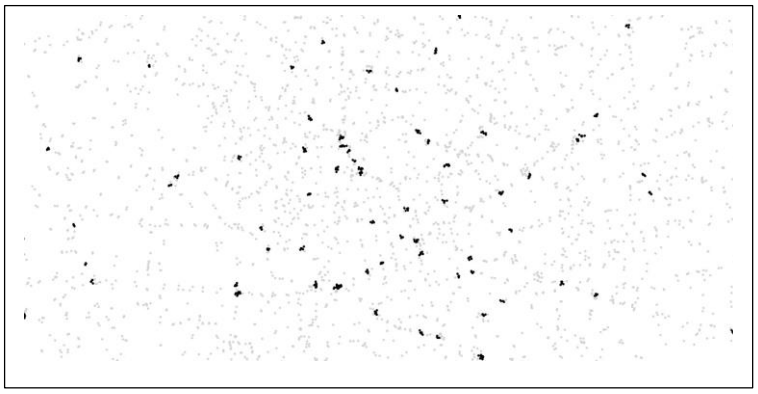
7.1.1. Finding the appropriate α value

In *scpt* algorithm, α determines the size of the clusters as well as the proximity of the vertices to the center of the clusters. Figure 7.7 shows the effects of varying α on the clustering results on the region designated by the orange rectangle in Figure 4.2.b (Paris dataset).

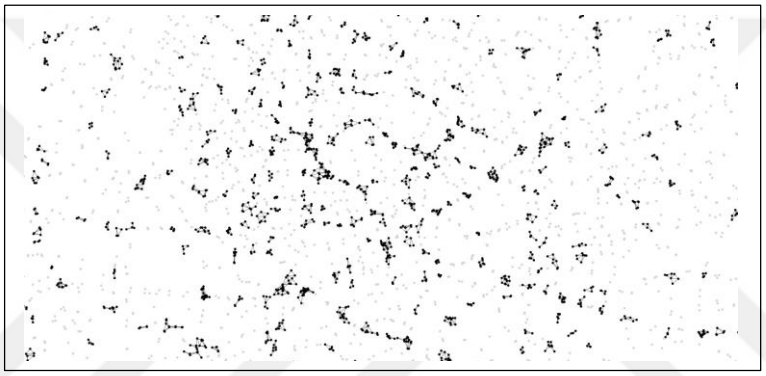
As α decreases the cluster sizes increases, however, there is no significant accumulations on one region, the even distribution of the clusters is preserved.

Table 7.3. presents the effects of varying α on all MPTN datasets. The table is divided into four parts, each part is dedicated to one MPTN dataset. In each part, the rows show similar statistics parameters presented in Table 7.2. In each network, as α decreases the number of clusters increases as well as the size of the clusters increases. On the other hand, as the last column shows that the percentages of the clusters larger than 100 vertices did not increase and stayed significantly small (0.1%).

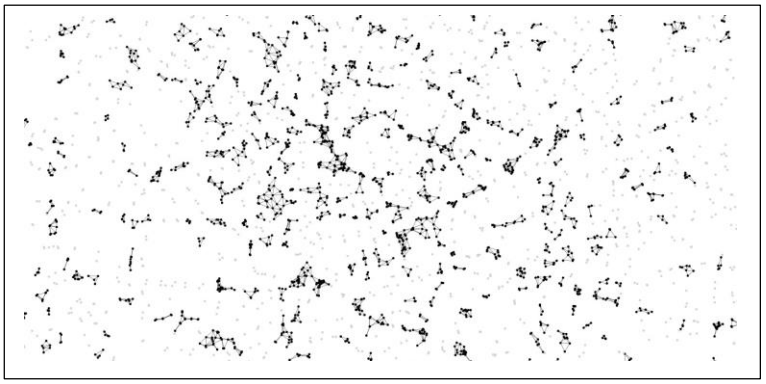
The distances from the nodes to the center of the cluster are as crucial as the size of the clusters for the hub location problem. The walking distance is usually assumed as 500 meters. Table 7.4. presents the distance statistics. The unit of the distance values is meters. The table is divided into six parts, in each part, the first and second rows show the means and standard deviations of all clusters, the third and fourth rows show the largest means and standard deviations obtained by *scpt*. As α gets smaller, the means and standard deviations increase, which is expected, since more distant nodes are included in the cluster by this choice. The statistics show that the popular assumption of 500 meters as walking distance to a hub is generally satisfied by $\alpha = 1/2$ even for the largest clusters. Because of that in most of the *scpt* experiments, this value is used. For example, on Sydney MPTN dataset (with $\alpha = 1/2$), the distance mean is approximately 82 meters with standard deviation of 33 meters. Also, the largest mean distance among all clusters of Sydney is 386 meters and the standard deviation of that cluster is 136 meters.



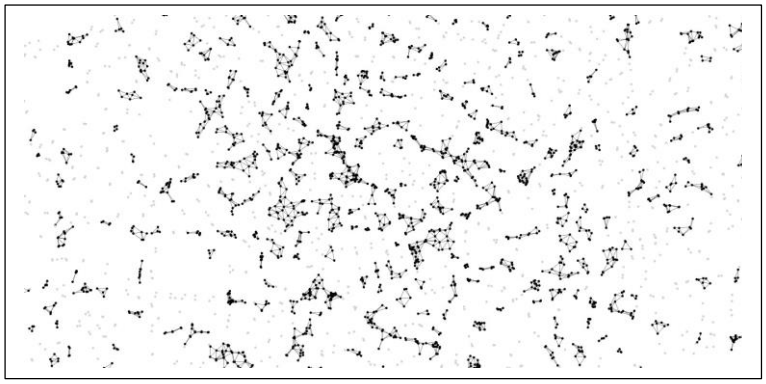
(a) α = 1



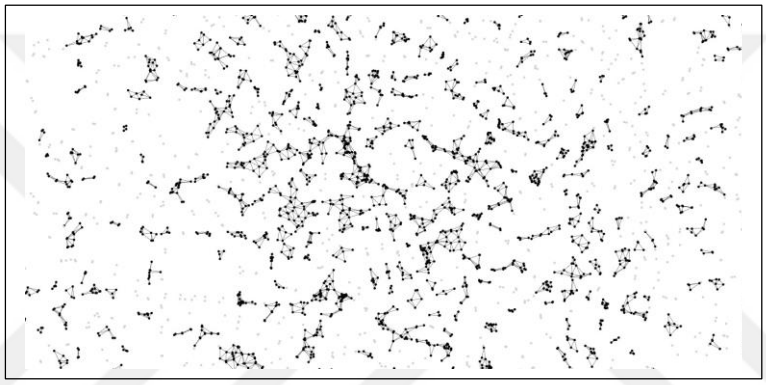
(b) α = 1/2



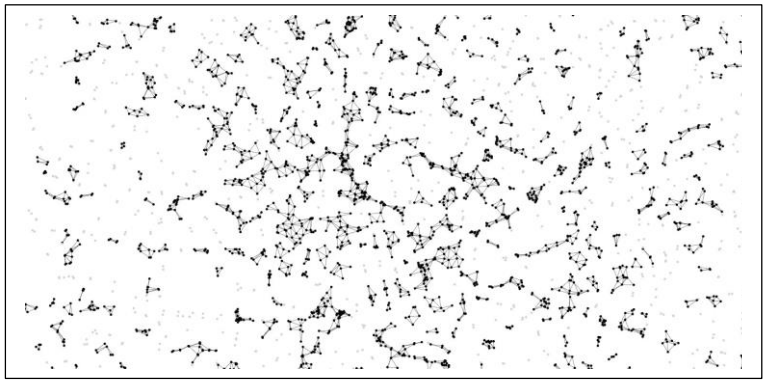
(c) α = 1/3



(d) α = 1/4



(e) α = 1/5



(f) α = 1/6

Figure 7.7. Effects of varying α values on scpt clusters.

Assuming that the distribution of the distances is close to normal distribution, further estimations can be done. Note that this assumption can easily be

justified because *scpt* removes long edges twice which imposes a normal distribution on the remaining edge distances. This assumption is especially true for large values of $\alpha = 1/2$. The values presented in the last rows of each section supports this idea. Except for Berlin dataset, even in the worst case (shown in the last row), only 6.3% of the nodes are more than 500 meters away from their cluster centers in the MPTN dataset of Brisbane. The decrease in α especially adversely effects the results of Brisbane. This is due to the structural irregularity in Brisbane MPTN dataset. Compared to the other datasets (except Berlin), the difference of node densities between the center and the borders of the city is significantly high. One can see from Table 7.4. that Berlin's dataset results are significantly higher-than the other cities. Each time α gets smaller by one degree, the distances from the center of the cluster gets bigger by almost 10%. This is due to spacious topology of Berlin city, among all the cities, it has the largest mean value (Definition 4) which is 136.67. Unlike the other datasets, even for $\alpha = 1/2$ the largest mean distance from the center of the cluster is bigger-than 500 meters, which is 610.34 meter and a standard deviation of that cluster is 331.00 meters.

Table 7.3. Effects of using different α values on *scpt* results.

α		Athens	Paris	Sydney	Brisbane	Berlin
1	Clusters	23	202	87	46	29
	Largest cl.	8	14	18	14	14
	mean	4.65	5.23	5.59	5.42	5.44
	stdev	1.06	1.82	2.69	2.39	2.39
	% cl. >100	-	-	-	-	-
	% ver. >100	-	-	-	-	-
1/2	Clusters	334	757	1485	595	184
	Largest cl.	29	41	60	147	24
	mean	5.55	6.06	5.61	5.96	6.15
	stdev	2.72	3.59	2.98	6.43	3.17
	% cl. >100	-	-	-	0.1%	-
	% ver. >100	-	-	-	4.1%	-
1/3	Clusters	467	871	1938	696	244
	Largest cl.	89	83	64	198	33
	mean	6.60	6.65	6.65	7.35	7.18
	stdev	5.32	5.06	4.28	8.97	4.89
	% cl. >100	-	-	-	0.1%	-
	% ver. >100	-	-	-	3.8%	-
1/4	Clusters	491	894	1999	699	257
	Largest cl.	133	83	68	198	40
	mean	7.45	7.15	7.43	8.22	8.02
	stdev	7.74	5.80	5.59	9.74	6.21
	% cl. >100	0.2%	-	-	0.1%	-
	% ver. >100	3.6%	-	-	3.4%	-
1/5	Clusters	485	914	1996	701	250
	Largest cl.	208	111	115	247	103
	mean	8.19	7.44	7.95	8.70	8.89
	stdev	11.10	6.63	6.90	11.87	9.29
	% cl. >100	0.2%	0.1%	0.05%	0.1%	0.4%
	% ver. >100	5.2%	1.6%	0.7%	4.0%	4.6%
1/6	Clusters	495	924	1976	688	247
	Largest cl.	208	131	130	247	104
	mean	8.40	7.65	8.31	9.23	9.41
	stdev	11.99	7.10	7.99	12.63	10.06
	% cl. >100	0.4%	0.1%	0.1%	0.1%	0.4%
	% ver. >100	7.6%	1.8%	1.5%	3.8%	4.4%

Table 7.4. Relation of α with the proximity of nodes to the centers of the clusters.

α		Athens	Paris	Sydney	Brisbane	Berlin
1	mean all cls.	26.93	25.10	21.15	19.43	122.20
	stdev all cls.	11.30	10.17	8.68	9.43	59.49
	lgst. mean	45.78	53.58	41.31	42.16	188.05
	lgst. stdev	24.42	27.66	24.15	18.03	65.50
	% cls. > 500m	0	0	0	0	0
1/2	mean all cls.	83.69	60.99	81.31	98.03	243.94
	stdev all cls.	34.84	25.11	32.69	38.21	106.66
	lgst. mean	223.35	216.40	386.31	357.64	610.34
	lgst. stdev	103.12	101.05	136.04	161.72	331.00
	% cls. > 500m	0	0	0	0.1%	11.4%
1/3	mean all cls.	109.96	80.69	111.24	135.27	294.43
	stdev all cls.	46.18	34.33	48.42	56.07	129.48
	lgst. mean	541.16	351.37	596.83	474.39	833.57
	lgst. stdev	241.95	141.28	342.99	229.49	434.93
	% cls. > 500m	0.4%	0.1%	0.7%	2.9%	23.7%
1/4	mean all cls.	125.33	92.68	128.61	155.28	327.50
	stdev all cls.	53.51	39.96	56.88	64.94	146.30
	lgst. mean	590.83	489.73	671.63	574.04	867.58
	lgst. stdev	251.85	290.66	391.47	303.54	438.14
	% cls. > 500m	1.2%	0.2%	1.6%	3.8%	31.9%
1/5	mean all cls.	136.71	100.47	139.46	166.14	352.08
	stdev all cls.	58.46	42.99	61.72	69.38	156.62
	lgst. mean	810.93	488.69	699.98	724.82	1733.73
	lgst. stdev	383.18	281.99	351.57	328.12	812.46
	% cls. > 500m	2.4%	0.5%	2.5%	4.9%	36.0%
1/6	mean all cls.	140.18	106.05	146.38	176.83	368.59
	stdev all cls.	59.61	45.30	65.38	74.89	162.70
	lgst. mean	814.42	501.86	754.67	818.32	1955.80
	lgst. stdev	818.32	292.09	302.15	390.25	825.91
	% cls. > 500m	2.6%	0.5%	3.1%	6.3%	40.0%

7.2 Case study: Greater London

TFL's Greater London MPTN dataset is used as input for *scpt*, 1005 clusters were generated. Figure 7.8 shows the distribution of the centroids of the clusters and zones (boroughs) on the map of Greater London.

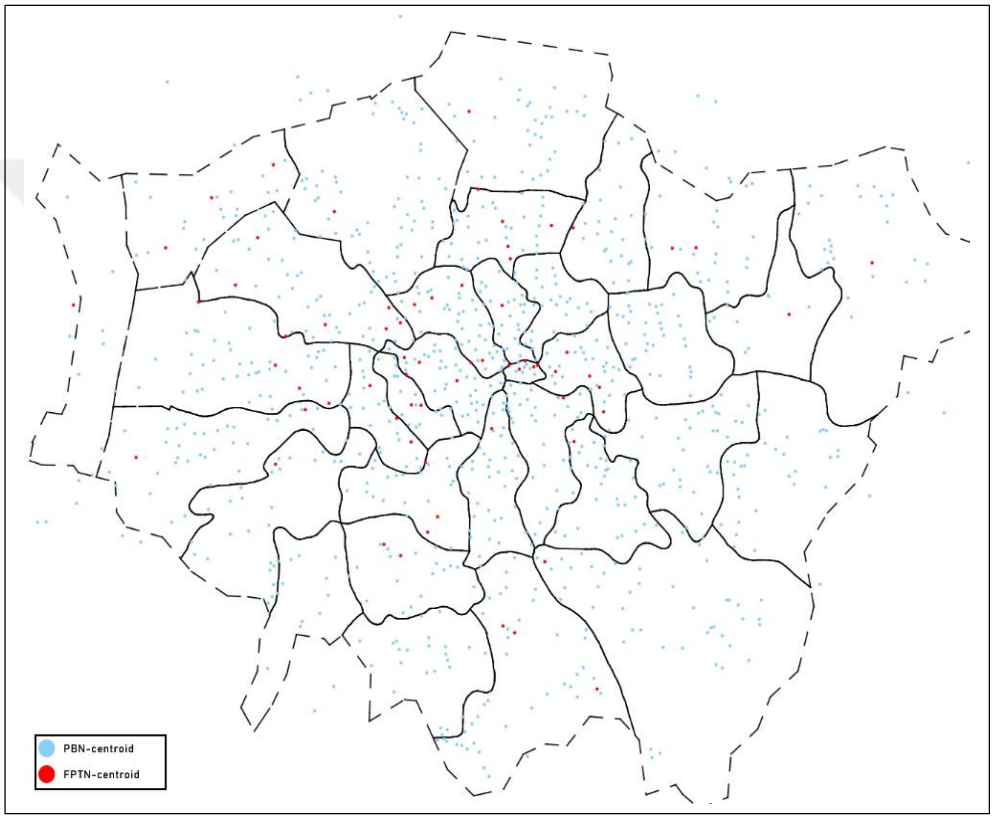


Figure 7.8. Distribution of the clusters and Boroughs on Grater London map.

7.2.1 Hub and spoke network results

The location of the hubs and their catchment area are shown in Figure 7.9. The cumulative direct coverage of all hubs is 94.52% of all centroids. The proposed method locates hubs as long as there are still FPTN-based hub candidates that are reachable by a single mode. If this condition is not met, PBN-based hubs are located to ensure maximum coverage of the network. The more hubs there are, the greater

the coverage of the network (B. Yu et al., 2013). As can be seen in Figure 7.9, even the hubs at the edge of the network overlap with at least two other hubs, providing passengers with better accessibility and more choice, maximizing the use of the network and leading to higher revenues. A total of 19 hubs were located, of which 17 are based on FPTN and 2 on PBN. Information about the hubs can be found in Table 7.5., in this table the hubs are sorted chronologically by location. The first column contains the NAPTAN (National Public Transport Access Nodes) id of the hub (NAPTAN: https://data.gov.uk/dataset/ff93ffc1-6656-47d8-9155-85ea0b8f2251/national-public-transport-access-nodes-naptan License: https://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/); the second column contains the common name by which the station is known. In this column, U.S. stands for Underground Station, R.S. for Rail Station and T.S. for Tram Station; columns 3 to 6 give the value of the criteria used in Equation (5). The last two hubs in this table are PBN-based and were not located using these criteria. Here, the demand and distance of the centroids in their catchment areas are given for comparison with the other hubs.

Initially, 21 FPTN-based hub lines were created; following the procedure to maximize network connectivity, the number of lines was increased to 40. The longest bus line in the TFL network is the N89 with a length of 36 km. This length was used as the maximum length for PBN-based hub lines. For PBN, 26 hub lines were created, here K = 10 is used for KSP. The value of K can be increased or decreased, which affects both the computation time of the algorithm and the search space. The larger the value of K, the longer the paths in the search space, which is undesirable due to the shortcomings of PBN such as low speed and traffic congestion. Using a small value of K ensures that all paths are close to the shortest path. The complete hub level network is shown in Figure 7.10. The hub line selection method proposed in this Thesis guarantees that passengers travel on the hub lines without making

transfer. As described in (Huang et al., 2018), the criteria for selecting hubs are to maximize demand coverage and distance to other hubs. In this Thesis, in addition to these criteria, multimodality and connectivity were also considered to maximize the probability of finding a single-mode line between two hubs. For this reason, there are candidates (e.g. 940GZZLUWRP, Table 7.5.) that serve only one line and one mode, but were still selected as hubs because they have suitable connectivity to existing hubs.

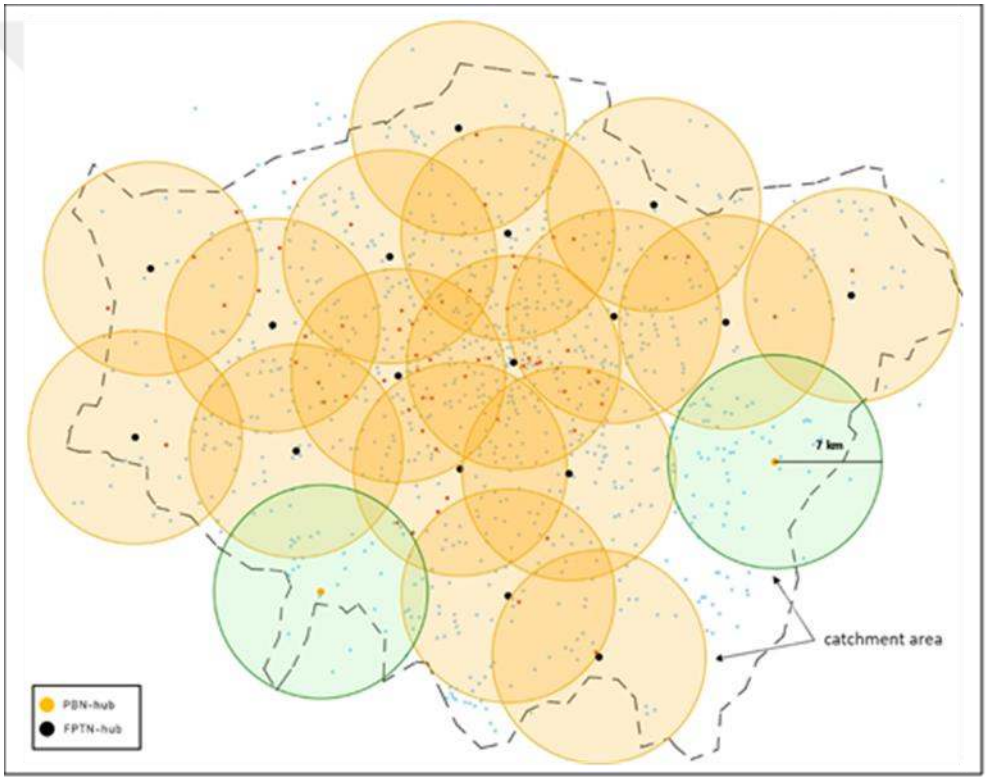


Figure 7.9. Hubs locations and network coverage.

Table 7.5. Hubs results.

Naptan ID	Common Name	C	M	DC	D
940GZZLUSTD	Stratford U. S.	9	4	309212	795.59
940GZZLUSPU	St. Paul's U. S.	2	1	462907	1328.25
940GZZLUCPS	Clapham South U. S.	2	1	342171	557.50
940GZZLUHPK	Holland Park U. S.	2	1	324426	903.59
940GZZLUPVL	Perivale U. S.	2	1	137014	262.80
940GZZLUWRP	West Ruislip U.	1	1	47826	61.68
940GZZLURMD	Richmond U. S.	1	2	143519	286.98
940GZZLUTPN	Turnpike Lane U. S.	2	1	219618	544.46
940GZZLUCKS	Cockfosters U. S.	1	1	75126	166.59
940GZZLUUPY	Upney U. S.	2	1	128388	243.88
940GZZLUHCH	Hornchurch U. S.	2	1	46170	117.87
940GZZLUBTX	Brent Cross U. S.	2	1	64302	450.51
940GZZLUHRC	Heathrow T. 2 & 3 U. S.	2	1	44327	150.59
910GHONROPK	Honor Oak Park R. S.	2	1	350107	360.02
910GWCROYDN	West Croydon R. S.	3	2	152586	374.45
940GZZCRNWA	New Addington T. S.	1	1	21039	99.53
940GZZLUWOF	Woodford U. S.	1	1	110131	234.71
490003975W	Geddes Place	-	-	27274	321.24
490014089N	Vincent Avenue	-	-	25864	300.98

Table 7.6. Allocation bus lines distances (km) results

Hub	Nbr. l.	Tot. dis.	Mean dis.	Stdev dis.	Short. l.	Long. l.
940GZZLUSTD	27	120.73	4.47	1.56	0.65	10.37
940GZZLUSPU	45	166.82	3.70	1.20	0.38	5.66
940GZZLUCPS	22	119.61	5.43	1.33	3.60	7.95
940GZZLUHPK	32	143.08	4.47	1.22	1.68	8.07
940GZZLUPVL	22	150.54	6.84	1.68	4.64	10.61
940GZZLUWRP	9	91.02	10.11	4.62	5.45	21.37
940GZZLURMD	20	118.78	5.93	1.39	3.12	8.28
940GZZLUTPN	21	99.32	4.72	2.15	1.67	13.13
940GZZLUCKS	20	131.90	6.59	2.53	2.77	14.12
940GZZLUUPY	15	148.08	9.87	6.85	2.07	20.98
940GZZLUHCH	13	97.40	7.49	2.88	3.14	14.26
940GZZLUBTX	23	130.40	5.66	1.70	2.99	9.82
940GZZLUHRC	14	92.67	6.61	1.95	2.76	10.43
910GHONROPK	28	171.92	6.14	1.77	2.82	11.30
910GWCROYDN	28	180.93	6.46	2.11	2.63	9.84
940GZZCRNWA	15	146.37	9.75	3.59	3.83	15.49
940GZZLUWOF	19	124.54	6.55	1.72	3.50	10.11
490003975W	28	202.78	7.24	3.20	0.57	13.94
490014089N	16	113.76	7.11	2.34	3.22	11.03

The allocation of bus lines is shown in Figure 7.11. A total of 417 bus lines were created. Table 7.6. presents the distance statistics of the bus line assigned to a hub. In this table, the first column contains the hub; the second column contains the number of lines assigned to the hub; the third column contains the sum of all line lengths in km; the fourth and fifth columns contain the mean and standard deviation of all line lengths; the sixth and seventh columns contain the shortest and longest line, respectively. As expected, the hub-and-spoke topology optimizes resource utilization. Compared to the TFL network, the proposed network has 233 fewer bus lines (including the PBN-based hub lines).

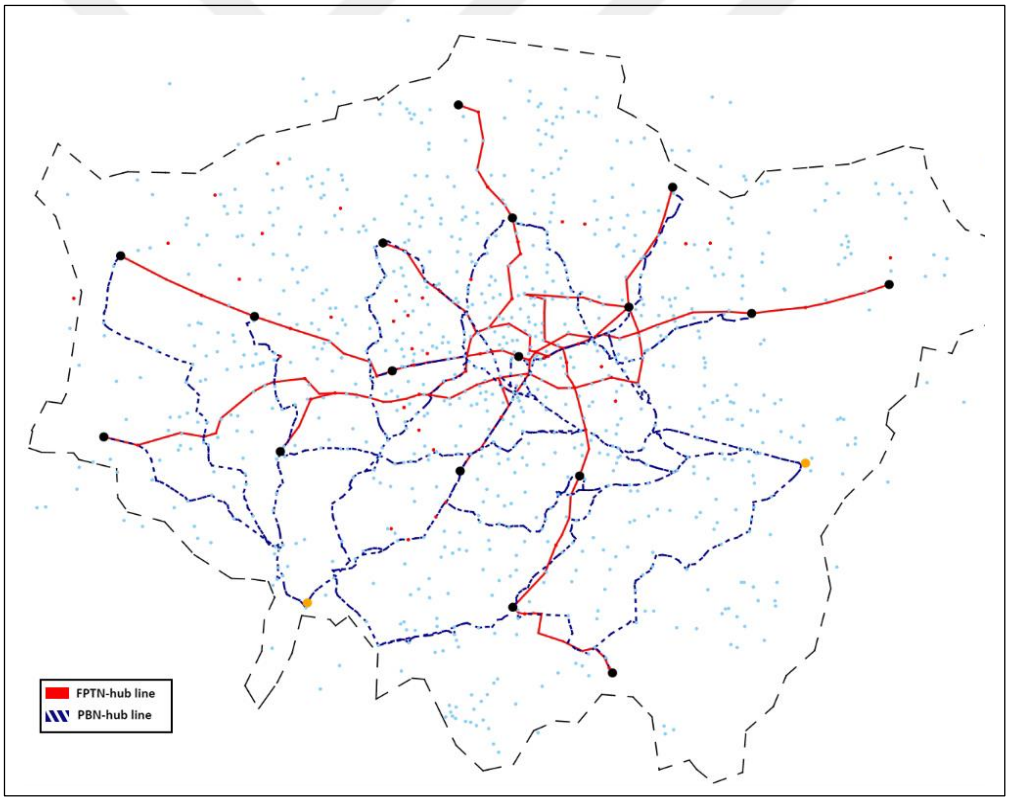


Figure 7.10. Hub-Level network

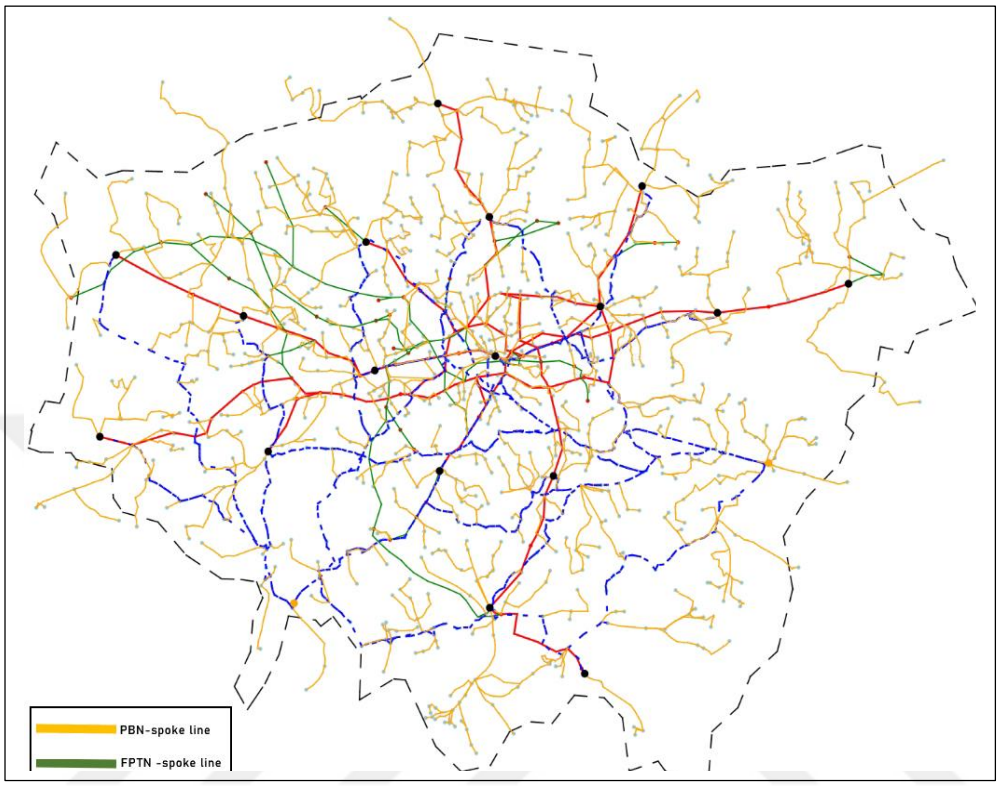


Figure 7.11. Hub-and-spoke network

7.2.2 Comparison Results

To show the effectiveness of the proposed MPTN design methodology, the results generated by the methodology presented in this study are compared with the results generated by the Journey API (JAPI) offered by TFL. The JAPI provides the ability to search for a journey by composing a multi-criteria query. For all JAPI queries, the aim is set to find the journey that takes the least time, allowing the use of walking and multimodality to optimize journey time. Only the modes used to create the proposed network are included in the query. In addition, a maximum transfer time of 10 minutes is set for each trip. The centroids generated by *scpt* are used as OD points for the trips. A trip is generated from each centroid to all other centroids. To reduce the number of queries to the JAPI, a symmetric matrix is used where one route is assumed to be equal to the opposite route. Due to COVID-19

restrictions, some routes have been temporarily shut down. For the JAPI query, a date and time is chosen when all routes are active. The exception is the Waterloo-Bank line, which has been closed until further notice. To allow a fair comparison, this line has also been removed from the proposed network.

TFL's Vision Zero action plan (Zero action plan: https://tfl.gov.uk/corporate/safety-and-security/road-safety/vision-zero-for-london) aims to have zero road deaths and serious injuries on London's transport network by 2041. One of the steps in this plan is to set safe speed limits. It has been determined that 20 mph (32.1869 km/h) is a safe speed and has been applied on several roads. Therefore, travel times for all bus routes in the proposed network are based on this speed limit. For the FPTN, the JAPI is used to determine travel times by searching for trips between adjacent stations.

Two experiments are conducted to generate trips with the proposed network; one with non-strict hubbing policy (NSHP) and the other with strict hubbing policy (SHP). NSHP is intended for direct comparison with JAPI. Here, the goal is to find the shortest trip, taking into account transfers between lines and walking distances to optimize travel time. DSP is used to generate the trip for NSHP. Naturally, a hub-and-spoke network is implemented to benefit from economies of scale. SHP is designed to show operators the performance of this topology. In SHP, trips are forced to travel via the hub. The exception is for trips where the OD centroids are on the same line. Here, a filtering algorithm is implemented that uses the relationship between the lines and vertices to generate the trips. In the case that the OD hubs are not adjacent, the DSP algorithm is used to find the shortest trip between the hubs. Although the single allocation method is used in this Thesis, the centroids located in the middle of a hub line are connected to at least two hubs. Since the search space is small in this case, the algorithm generates all possible trips and returns the shortest trip.

For simplicity, in NSHP and SHP, the transfer time, including hub-level transfer, is set to 5 minutes. In addition, it is assumed that appropriate schedule and frequency are provided. For each hub line a discount factor of 0.9 was applied.

A total of 504510 queries were sent to JAPI, of which 47 trips were excluded due to negative responses. The same trips were excluded from NSHP and NHP.

The sum of all trips generated by JAPI is 672298.31 hours, for NSHP the sum of trips is 501311.14 hours, which is an improvement of 25.43% compared to JAPI. For SHP, the sum of trips is 558640.62 hours, an improvement of 16.90% compared to JAPI.

Table 7.7. contains the data and statistics for trips produced by JAPI, NSHP and SHP. The first column of the table shows the duration of the trips. Each row has a span of 5 minutes, except for the last three rows where the span is 30 minutes for brevity. Each solution has three columns: The first column shows the number of trips whose duration falls within the specified range, and the second and third columns show the mean and standard deviation of all these trips.

Figure 7.12 shows the comparison of the number of transfers included in the trips. NSHP and SHP have slightly more trips without transfers than JAPI, because much of the network is covered by FPTN, which offers several direct trips. It is clear that SHP has more trips with one and two transfers than JAPI and NSHP as the transfers are forced in SHP. In addition, 88.96% of the trips generated for SHP also include a maximum of four transfers. In contrast, NSHP and JAPI have only 61.25% and 67.76% of trips with a maximum of four transfers respectively.

In general, SHP is the strategy used by operators for hub-and-spoke networks. Traffic flow is concentrated on the hub lines to benefit from economies of scale. Therefore, only the performance of the proposed network with SHP is considered in this part. SHP not only outperforms JAPI in terms of total travel time, but also provides 32.51% more trips under 60 minutes (see Table 7.6). This is because the proposed network focuses on creating lines that connect centroids, which increases the number of direct trips, especially when the centroids are close to each

other. In contrast, in the TFL network, these centroids can belong to different lines. This also explains why SHP has fewer trips with transfers greater than or equal 4 than JAPI.

Since hub-and-spoke topology is presented as a solution for the MPTN design, here some statistics on the mode interconnection for the trips generated with SHP are given. 93.99% of the trips include a spoke bus line; 75.24% include FPTN hub lines; 40.11% include PBN hub lines; and 4.58% include FPTN spoke lines.

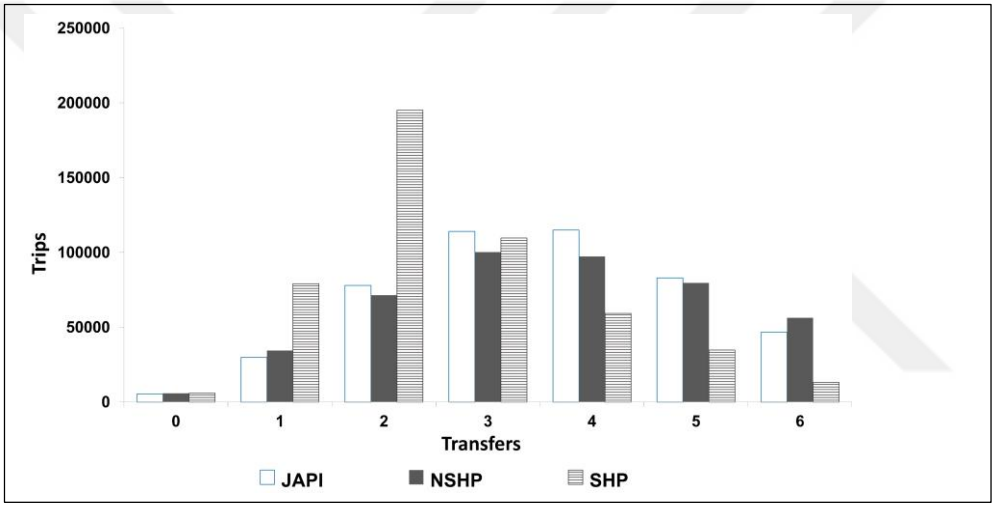


Figure 7.12. Transfers Comparison

Table 7.7. Trips statistics

TT	Trips	Mean	Stdev	Trips	Mean	Stdev	Trips	Mean	Stdev
<=5	945	2.96	1.36	2024	2.77	1.28	1143	2.71	1.36
5-10	1020	8.19	1.40	3167	7.90	1.44	1748	7.83	1.45
10-15	2253	13.30	1.36	7151	12.74	1.42	6511	13.01	1.36
15-20	4268	18.22	1.37	11791	17.67	1.41	13605	17.64	1.41
20-25	6983	23.19	1.39	15885	22.61	1.43	14258	22.45	1.45
25-30	10173	28.14	1.42	21597	27.36	1.43	16626	27.67	1.44
30-35	13594	33.09	1.41	27334	32.58	1.44	25359	32.63	1.42
35-40	17298	38.08	1.41	32601	37.55	1.44	28784	37.47	1.44
40-45	20968	43.07	1.41	35174	42.50	1.44	27837	42.52	1.44
45-50	24142	48.03	1.40	36663	47.50	1.43	30467	47.53	1.44
50-55	26703	53.02	1.40	36995	52.50	1.44	32775	52.51	1.44
55-60	28439	58.01	1.41	36774	57.50	1.44	33208	57.52	1.44
60-90	172087	75.17	8.53	172464	73.21	8.40	169730	73.58	8.43
90-120	113766	103.88	8.48	58008	101.24	7.98	70819	102.13	8.48
> 120	61827	141.96	22.66	6841	129.22	8.27	31510	142.81	17.73

8 CONCLUSION

This study presents a two-phase optimization approach that generates a hub-and-spoke network for MPTN. In the first phase, a spatial clustering algorithm (*scpt*) specifically designed for MPTN is presented. The algorithm requires no initial setup procedure and adapts smoothly to different MPTN architectures. *scpt* is tested on real MPTNs of the cities of Athens, Paris, Sydney and Brisbane. Our tests show that even the largest network (Sydney, 24063 nodes) is processed in less than a minute. The algorithm does not require any special parameters to be set depending on the network. There is a single parameter (α) that can be used to choose the size of the clusters, and it produces similar results for all networks. The algorithm can be easily used with any optimization method and can simplify the complexity of other steps of MPTN design problems. In the second phase, an algorithm based on multicriteria decision making is presented that simultaneously locates hubs and hub lines.

The complete hub-and-spoke network is based on the clusters generated by *scpt* and the FPTN network. The proposed methodology finds the required number of hubs to ensure maximum coverage of the network while optimizing the use of resources. The proposed methodology has been tested on Greater London's MPTN. The resulting hub-and-spoke network was used to generate trips under strict hubbing policy and compared with the trips generated by the Journey API provided by TFL. The results show that the proposed network improves the overall travel time and number of transfers while using fewer lines than the original network.

In a future work, various aspects of the proposed methodology can be improved. In this Thesis, the operating costs of hubs and hub lines is not considered; this could be included as a criterion in the decision model. Also, a very strict discount value for the hub lines (0.9) is used. The determination of the appropriate discount value can be explored in depth. In addition, it is assumed that there is an appropriate schedule and frequency. Optimization of these parameters can extend this research work in the future.

REFERENCES

Abbaspour, R., & Samadzadegan, F. (2010). An evolutionary solution for multimodal shortest path problem in metropolises. *Computer Science and Information Systems*, *7*(4), 789–811. https://doi.org/10.2298/CSIS090710024A

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, 94–105.

Ahmed, L., Mumford, C., & Kheiri, A. (2019). Solving urban transit route design problem using selection hyper-heuristics. *European Journal of Operational Research*, *274*(2), 545–559. https://doi.org/10.1016/j.ejor.2018.10.022

Alumur, S., & Kara, B. Y. (2008). Network hub location problems: The state of the art. *European Journal of Operational Research*, *190*(1), 1–21.

Ankerst, M., Breunig, M. M., Kriegel, H.-P., & Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. *ACM Sigmod Record*, *28*(2), 49–60.

Bastian, Mathieu, Sebastien Heymann, & Mathieu Jacomy. (2009). Gephi: an open source software for exploring and manipulating networks. *Proceedings of the International AAAI Conference on Web and Social Media*, *3*(1), 361–362.

Beltran, B., Carrese, S., Cipriani, E., & Petrelli, M. (2009). Transit network design with allocation of green vehicles: A genetic algorithm approach. *Transportation Research Part C: Emerging Technologies*, *17*(5), 475–483. https://doi.org/10.1016/j.trc.2009.04.008

Bielli, M., Boulmakoul, A., & Mouncif, H. (2006). Object modeling and path computation for multimodal travel systems. *European Journal of Operational Research*, *175*(3), 1705–1730. https://doi.org/10.1016/j.ejor.2005.02.036

Carbonneaux, Y., Laborde, J.-M., & Madani, R. M. (1996). *CABRI-Graph: A tool for research and teaching in graph theory* (pp. 123–126). https://doi.org/10.1007/BFb0021796

Carpenter, G. A., & Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, *37*(1), 54–115.

Ceder, A., & Wilson, N. H. M. (1986). Bus network design. *Transportation Research Part B: Methodological*, *20*(4), 331–344. https://doi.org/10.1016/0191-2615(86)90047-0

Cipriani, E., Gori, S., & Petrelli, M. (2012). Transit network design: A procedure and an application to a large urban area. *Transportation Research Part C: Emerging Technologies*, *20*(1), 3–14. https://doi.org/10.1016/j.trc.2010.09.003

CIPRIANI, E., PETRELLI, M., & FUSCO, G. (2006). A multimodal transit network design procedure for urban areas. *Advances in Transportation Studies*, *10*, 5–20.

Computational Geometry. (2008). In *Computational Geometry* (pp. 1–17). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-77974-2_1

Daganzo, C. F. (2010). Structure of competitive transit networks. *Transportation Research Part B: Methodological*, *44*(4), 434–446.

Dalkılıç, F., Doğan, Y., Birant, D., Kut, R. A., & Yılmaz, R. (2017). A Gradual Approach for Multimodel Journey Planning: A Case Study in Izmir, Turkey. *Journal of Advanced Transportation*, *2017*, 1–14. https://doi.org/10.1155/2017/5656323

Delaunay, B. (1934). Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, *7*(793–800).

Deng, M., Liu, Q., Cheng, T., & Shi, Y. (2011). An adaptive spatial clustering algorithm based on Delaunay triangulation. *Computers, Environment and Urban Systems*, *35*(4), 320–332.

der Merwe, D. W., & Engelbrecht, A. P. (2003). Data clustering using particle swarm optimization. *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, *1*, 215–220.

Developers, C. M. (2020). *Apache Commons Math, Release 3.6*.

Dib, O., Dib, M., & Caminada, A. (2018). Computing Multicriteria Shortest Paths in Stochastic Multimodal Networks Using a Memetic Algorithm. *International Journal on Artificial Intelligence Tools*, *27*(07), 1860012. https://doi.org/10.1142/S0218213018600126

Dib, O., Moalic, L., Manier, M.-A., & Caminada, A. (2017). An advanced GA–VNS combination for multicriteria route planning in public transit networks. *Expert Systems with Applications*, *72*, 67–82. https://doi.org/10.1016/j.eswa.2016.12.009

Ellson, J., Gansner, E., Koutsofios, L., North, S. C., & Woodhull, G. (2002). *Graphviz— Open Source Graph Drawing Tools* (pp. 483–484). https://doi.org/10.1007/3-540-45848-4_57

Erich, S., & Arthur, Z. (2019). ELKI: A large open-source library for data analysis ELKI Release 0.7.5 "Heidelberg." *ArXiv Preprint ArXiv:1902.03616*.

Ernst, A. T., & Krishnamoorthy, M. (1996). Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science*, *4*(3), 139–154.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). Density-based spatial clustering of applications with noise. *Int. Conf. Knowledge Discovery and Data Mining*, *240*, 6.

Estivill-Castro, V., & Lee, I. (2000). Autoclust: Automatic clustering via boundary extraction for mining massive point-data sets. *In Proceedings of the 5th International Conference on Geocomputation*.

Fan, W., & Machemehl, R. B. (2006a). Optimal Transit Route Network Design Problem with Variable Transit Demand: Genetic Algorithm Approach. *Journal*

of Transportation Engineering, *132*(1), 40–51. https://doi.org/10.1061/(ASCE)0733-947X(2006)132:1(40)

Fan, W., & Machemehl, R. B. (2006b). Using a Simulated Annealing Algorithm to Solve the Transit Route Network Design Problem. *Journal of Transportation Engineering*, *132*(2), 122–132. https://doi.org/10.1061/(ASCE)0733-947X(2006)132:2(122)

Fan, W., & Machemehl, R. B. (2008). Tabu Search Strategies for the Public Transportation Network Optimizations with Variable Transit Demand. *Computer-Aided Civil and Infrastructure Engineering*, *23*(7), 502–520. https://doi.org/10.1111/j.1467-8667.2008.00556.x

Farahani, R. Z., Hekmatfar, M., Arabani, A. B., & Nikbakhsh, E. (2013). Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering*, *64*(4), 1096–1109. https://doi.org/10.1016/j.cie.2013.01.012

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*(2), 139–172.

FORTUNE, S. (1995). *VORONOI DIAGRAMS and DELAUNAY TRIANGULATIONS* (pp. 225–265). https://doi.org/10.1142/9789812831699_0007

Fournier-Viger, P., Lin, J. C.-W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., & Lam, H. T. (2016). The SPMF open-source data mining library version 2. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 36–40.

Gallo, G., & Pallottino, S. (1988). Shortest path algorithms. *Annals of Operations Research*, *13*(1), 1–79. https://doi.org/10.1007/BF02288320

Gallo, M., D'Acierno, L., & Montella, B. (2011). *A multimodal approach to bus frequency design*. 193–204. https://doi.org/10.2495/UT110171

Gastner, M. T., & Newman, M. E. J. (2006). Shape and efficiency in spatial distribution networks. *Journal of Statistical Mechanics: Theory and Experiment*, *2006*(01), P01015.

Gelareh, S., & Nickel, S. (2008). A benders decomposition for hub location problems arising in public transport. In *Operations Research Proceedings 2007* (pp. 129–134). Springer.

Guha, S., Rastogi, R., & Shim, K. (2000). ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, *25*(5), 345–366.

Guibas, L. J., Knuth, D. E., & Sharir, M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, *7*(1–6), 381–413. https://doi.org/10.1007/BF01758770

Hagberg, Aric, Pieter Swart, & Daniel S Chult. (2008). Exploring network structure, dynamics, and function using NetworkX. *Los Alamos National Lab.(LANL), Los Alamos, NM (United States)*.

Haklay, M., & Weber, P. (2008). OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, *7*(4), 12–18. https://doi.org/10.1109/MPRV.2008.80

Handl, J., & Meyer, B. (2007). Ant-based and swarm-based clustering. *Swarm Intelligence*, *1*(2), 95–113.

Hinneburg, A., Keim, D. A., & others. (1998). *An efficient approach to clustering in large multimedia databases with noise*.

Huang, D., Liu, Z., Fu, X., & Blythe, P. T. (2018). Multimodal transit network design in a hub-and-spoke network framework. *Transportmetrica A: Transport Science*, *14*(8), 706–735.

Karypis, G., Han, E.-H., & Kumar, V. (1999). Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, *32*(8), 68–75.

Kujala, R., Weckström, C., Darst, R. K., Mladenović, M. N., & Saramäki, J. (2018). A collection of public transport network data sets for 25 cities. *Scientific Data*, *5*, 180089.

Lee, D. T., & Schachter, B. J. (1980). Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences*, *9*(3), 219–242. https://doi.org/10.1007/BF00977785

Liu, D., Nosovskiy, G. v, & Sourina, O. (2008). Effective clustering and boundary detection algorithm based on Delaunay triangulation. *Pattern Recognition Letters*, *29*(9), 1261–1273.

Liu, L., Yang, J., Mu, H., Li, X., & Wu, F. (2014). Exact algorithms for multi-criteria multi-modal shortest path with transfer delaying and arriving time-window in urban transit network. *Applied Mathematical Modelling*, *38*(9–10), 2613–2629. https://doi.org/10.1016/j.apm.2013.10.059

Liu, Z., Chen, X., Meng, Q., & Kim, I. (2018). Remote park-and-ride network equilibrium model and its applications. *Transportation Research Part B: Methodological*, *117*, 37–62. https://doi.org/10.1016/j.trb.2018.08.004

López, D., & Lozano, A. (2020). Shortest hyperpaths in a multimodal hypergraph with real-time information on some transit lines. *Transportation Research Part A: Policy and Practice*, *137*, 541–559. https://doi.org/10.1016/j.tra.2019.09.020

Lozano, A., & Storchi, G. (2001). Shortest viable path algorithm in multimodal networks. *Transportation Research Part A: Policy and Practice*, *35*(3), 225–241. https://doi.org/10.1016/S0965-8564(99)00056-7

MacQueen, J., & others. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, *1*(14), 281–297.

Majima, T., Takadma, K., Watanabe, D., & Katuhara, M. (2017). Generating hub-spoke network for public transportation: comparison between genetic algorithm and cuckoo search algorithm. In *Intelligent and Evolutionary Systems* (pp. 263–275). Springer.

Michail, D., Kinable, J., Naveh, B., & Sichi, J. v. (2020). JGraphT—A Java Library for Graph Data Structures and Algorithms. *ACM Transactions on Mathematical Software*, *46*(2), 1–29. https://doi.org/10.1145/3381449

Ng, R. T., & Han, J. (1994). Efficient and Effective clustering methods for spatial data mining. *Proceedings of VLDB*, 144–155.

Ng, R. T., & Jiawei Han. (2002). CLARANS: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, *14*(5), 1003–1016. https://doi.org/10.1109/TKDE.2002.1033770

Nickel, S., Schöbel, A., & Sonneborn, T. (2001). Hub location problems in urban traffic networks. In *Mathematical methods on optimization in transportation systems* (pp. 95–107). Springer.

O'kelly, M. E. (1987). A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, *32*(3), 393–404.

Park, H.-S., & Jun, C.-H. (2009). A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, *36*(2), 3336–3341.

Peng, Y., Mo, Z., & Liu, S. (2021). Passenger's routes planning in stochastic common-lines' multi-modal transportation network through integrating Genetic Algorithm and Monte Carlo simulation. *Archives of Transport*, *59*(3), 73–92. https://doi.org/10.5604/01.3001.0015.0123

Rasmussen, C. E. (2000). The infinite Gaussian mixture model. *Advances in Neural Information Processing Systems*, 554–560.

Rodriguez, A., & Laio, A. (2014). Clustering by fast search and find of density peaks. *Science*, *344*(6191), 1492–1496.

Rostami, M Ali and Azadi, Azin and Seydi, & Masoumeh. (2014). Graphtea: Interactive graph self-teaching tool. *Proc. 2014 Int. Conf. Edu. \& Educat. Technol. II*, 48–52.

Santos, G., Maoh, H., Potoglou, D., & von Brunn, T. (2013). Factors influencing modal split of commuting journeys in medium-size European cities. *Journal of*

Transport Geography, *30*, 127–137. https://doi.org/10.1016/j.jtrangeo.2013.04.005

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)*, *42*(3), 1–21.

Sheikholeslami, G., Chatterjee, S., & Zhang, A. (1998). Wavecluster: A multi-resolution clustering approach for very large spatial databases. *VLDB*, *98*, 428–439.

Sinclair, D. (2016). S-hull: a fast radial sweep-hull routine for Delaunay triangulation. *ArXiv Preprint ArXiv:1604.01428*.

Song, Z., He, Y., & Zhang, L. (2017). Integrated planning of park-and-ride facilities and transit service. *Transportation Research Part C: Emerging Technologies*, *74*, 182–195. https://doi.org/10.1016/j.trc.2016.11.017

Su, P., & Scot Drysdale, R. L. (1997). A comparison of sequential Delaunay triangulation algorithms. *Computational Geometry*, *7*(5–6), 361–385. https://doi.org/10.1016/S0925-7721(96)00025-9

Sung, C. S., & Jin, H. W. (2001). Dual-based approach for a hub network design problem under non-restrictive policy. *European Journal of Operational Research*, *132*(1), 88–105.

Vafaei, N., Ribeiro, R. A., & Camarinha-Matos, L. M. (2018). Selection of normalization technique for weighted average multi-criteria decision making. *Doctoral Conference on Computing, Electrical and Industrial Systems*, 43–52.

Wan, Q. K., & Lo, H. K. (2009). Congested multimodal transit network design. *Public Transport*, *1*(3), 233–251. https://doi.org/10.1007/s12469-009-0015-8

Wang, W., Yang, J., Muntz, R., & others. (1997). STING: A statistical information grid approach to spatial data mining. *VLDB*, *97*, 186–195.

Wang, Z., & Chen, Y. (2012). Development of location method for urban public transit networks based on hub-and-spoke network structure. *Transportation Research Record*, *2276*(1), 17–25.

Xu, D., & Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, *2*(2), 165–193.

Yu, B., Zhu, H., Cai, W., Ma, N., Kuang, Q., & Yao, B. (2013). Two-phase optimization approach to transit hub location–the case of Dalian. *Journal of Transport Geography*, *33*, 62–71.

Yu, J., Liu, Y., Chang, G.-L., Ma, W., & Yang, X. (2009). Cluster-based hierarchical model for urban transit hub location planning: formulation, solution, and case study. *Transportation Research Record*, *2112*(1), 8–16.

Yu, J., Liu, Y., Chang, G.-L., Ma, W., & Yang, X. (2011). Locating urban transit hubs: Multicriteria model and case study in China. *Journal of Transportation Engineering*, *137*(12), 944–952.

Yu, J., Liu, Y., Chang, G.-L., & Yang, X.-G. (2008). Cluster-based optimization of urban transit hub locations: Methodology and case study in China. *Transportation Research Record*, *2042*(1), 109–116.

Yuan, Y., & Yu, J. (2018). Locating transit hubs in a multi-modal transportation network: A cluster-based optimization approach. *Transportation Research Part E: Logistics and Transportation Review*, *114*, 85–103.

Zhan, Q., Deng, S., & Zheng, Z. (2017). An adaptive sweep-circle spatial clustering algorithm based on gestalt. *ISPRS International Journal of Geo-Information*, *6*(9), 272.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *ACM Sigmod Record*, *25*(2), 103–114.