UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO (COMPUTER SCIENCE)

BRUNO COSWIG FISS

Exact and Metaheuristic Algorithms for the Urban Transit Routing Problem

Final Report presented in partial fulfillment of the requirements for the degree of Bachelor of Computer Science

Prof. Dr. Marcus Rolf Peter Ritt
Advisor

Porto Alegre, July 2012

CIP – CATALOGING-IN-PUBLICATION

Fiss, Bruno Coswig

Exact and Metaheuristic Algorithms for the Urban Transit Routing Problem / Bruno Coswig Fiss. – Porto Alegre: PPGC da UFRGS, 2012.

69 f.: il.

Final Report (Bachelor) – Universidade Federal do Rio Grande do Sul. Ciência da Computação (Computer Science), Porto Alegre, BR–RS, 2012. Advisor: Marcus Rolf Peter Ritt.

1. Urban Transit Routing Problem. 2. Mixed Integer Programming. 3. Metaheuristics. I. Ritt, Marcus Rolf Peter. II. Título.

“The real voyage of discovery consists not in seeking new lands, but seeing with new eyes.”
— MARCEL PROUST

ACKNOWLEDGEMENTS

I would like to thank my advisor, professor Marcus Ritt, whose support, effort and knowledge were essential during the whole work.

I am also grateful for my colleagues and professors, who always encouraged me to strive for knowledge and attempt new challenges.

At last, and most importantly, I would like to thank my family. Besides always being there for me, they taught me the most important concepts in life, including honesty, friendship and morality.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ATT	Average Travel Time
DFS	Depth First Search
GA	Genetic Algorithm
MIP	Mixed Integer Programming
OD	Origin-Destination
TTT	Total Travel Time
UTNDP	Urban Transit Network Design Problem
UTRP	Urban Transit Routing Problem
UTSP	Urban Transit Scheduling Problem

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

The urban transit routing problem (UTRP) consists of finding satisfying routes for public transportation within a city or region. Urban scenarios get more complex as time goes by, making the design of routes an overwhelming task whose results are often unsatisfactory, with high costs and travel times. We develop an exact MIP formulation for the problem and obtain best solutions, which were previously unknown, for common benchmarks. We also develop a multi-objective genetic algorithm to solve thie problem with higher quality and more efficiently than with current techniques. We benchmark our solutions on generally available real and artificial test cases and achieve better results.

Keywords: Urban Transit Routing Problem, Mixed Integer Programming, Metaheuristics.

Algoritmos exatos e metaheurísticos para o Problema de Roteamento de Transporte Urbano

RESUMO

O problema de roteamento de transporte urbano consiste em encontrar rotas satisfatórias para transporte público em uma cidade ou região. Cenários urbanos se tornam mais complexos com o passar do tempo, tornando o planejamento de rotas uma tarefa proibitivamente difícil, cujos resultados são frequentemente insatisfatórios, com altos custos e tempos de viagem. Nós propomos uma formulação MIP exata para o problema e obtemos resultados ótimos, que até então não eram conhecidos, para casos de teste usados na literatura. Nós também desenvolvemos um algoritmo genético multiobjetivo para resolver o problema com mais qualidade e eficiência do que com técnicas atuais. Testamos nossas soluções com cenários reais e artificiais publicados anteriormente e obtemos resultados superiores.

Palavras-chave: Problema de Roteamento Urbano, Programação Inteira Mista, Metaheurísticas.

1 INTRODUCTION

Public transportation is of vital importance in almost every place in the world. It shrinks distances and allows a population to interact and produce more. In comparison to private transportation, it utilizes less space, allowing more people to benefit from it, preventing jams, and is less harmful to the environment.

But, traffic is growing substantially, even in places where the number of people is stable. This creates complex scenarios, with lots of demand that must be fulfilled, making an efficient public transport system mostly desirable. The task of designing a public transit network is commonly performed by traffic engineers and specialists, planners that attempt to satisfy passengers, or generate profit, while also attending stake-holders wishes, such as government, businesses and taxpayers.

This complicated task can surely benefit from computer assistance. The study of urban transit network design under the name Urban Transit Network Design Problem, the UTNDP, aims at obtaining good knowledge, tools and techniques for the design of public transport systems, observing the mentioned restrictions and demands. It is an **NP-hard** problem with multiple criteria. Because of its complexity, this problem is commonly divided in two parts: the Urban Transit Routing Problem (UTRP) and the Urban Transit Scheduling Problem (UTSP) (CHAKROBORTY, 2004).

The UTRP concerns deciding routes where public traffic will flow in such a way that it can provide good service (short travel times and low number of transfers) for passengers, with an acceptable cost and satisfying stake-holder demands. It is of vital importance in public transport systems since it directly defines the routes taken by city dwellers on their daily tasks. The UTSP decides the exact schedule of vehicles in a given set of routes so that the demand at each time period is satisfied, also taking many restrictions into consideration. Because the step that requires most planning is the UTRP, since changes to routes are normally harder to implement, and of the complexity of the problem, we focus only on the UTRP in this work.

Our main goal in this work is to create, implement and describe algorithms to solve the UTRP. We do not intend to create a tool or a fully ready to use application for specific traffic scenarios, but rather to develop successful algorithms and techniques for the problem in general, which can later prove useful for other, more specialized real life applications.

We notice, as is explained in Section 1.1, that thorough comparison of techniques is not commonly done, since most works adapt problem definitions to a scenario, with new constraints or objective functions. But since problem statements differ, comparison of

solutions is not possible. Another frequent issue is the lack of common available and used benchmarks for the problem, necessary for good comparisons.

Comparison is but very important. With a well performed evaluation, specialists and tool developers will possess better knowledge for the selection of the right algorithms and techniques, and will thus make better informed decisions for the UTRP and the UTNDP.

Therefore, we use a simple and generic problem definition that is compatible with many previous works. Our definition is based on the definition used by Lang Fan *et al.* (2009), and is described in detail in Chapter 2.

After formally defining the problem, we give two solutions to it. The first one is based on Mixed Integer Programming, and solves the problem completely, without human aid. To the best of our knowledge, no full multi-objective linear model for UTRP has been published before. Exact solutions are often used to optimize one or two parameters in an existing network, or for small-sized networks. It has even been said that Mixed Integer Programming cannot model the UTRP (CHAKROBORTY, 2004). We prove it to be possible, and obtain optimal results for a well known benchmark, to be described in Section 1.1. We also propose two applications for the model when dealing with larger networks: using it in a *divide-and-conquer* fashion, handling small groups of nodes at each time, until the optimization of the whole network is ready, and using the linear relaxation of the model to obtain scores between 0 and 1 that characterize how important an edge is, information that can then be used in the generation of routes in another approach. Our formulation is described in Chapter 3.

The second solution is a genetic algorithm. Heuristics and metaheuristics are the common choice for solving the UTRP (ÁLVAREZ et al., 2010), given the problem complexity and high number of constrains. We base our solution on a genetic algorithm since previous works were able to achieve good results with it. We take advantage of key aspects that were not used before, and attempt to achieve a more effective algorithm by:

- carefully selecting initial solutions from many different sources, including MIP relaxation, minimum spanning tree, shortest paths and from greedy algorithms;
- applying *simplification* to prevent unnecessary routes;
- using operators such as *exchange* and *crossover*, that exchange characteristics between routes and route sets;
- not letting a feasible solution be removed if it is *undominated* (term which is defined in Section 2.2.1 and by allowing different route set sizes in the same population.

We present the whole algorithm in detail, allowing a reader to reproduce our experiments completely. An important step that is sometimes not well documented (FAN; MUMFORD; EVANS, 2009) is how the initial population is created. In Chapter 4, this and all other involved procedures are described in detail.

Finally, to assess the quality of our solutions, we compare our work with previous ones that use a common benchmark to be described in Section 1.1. The results are listed and analyzed in Chapter 5.

Finally, our conclusions and final remarks are then presented in Chapter 6.

1.1 Related Work

Given our goal of developing successful algorithms and techniques for the UTRP, we are mostly focused on studying generic problem definitions and solutions that allow fair comparison of results and thus make it possible to rate the used algorithms and techniques. Furthermore, since this work follows two main solution ideas, namely mathematical programming and metaheuristic (or, more specifically, *evolutive*), we analyze the literature in these areas.

One of the first attempts to tackle the UTNDP generically, and with computer aid, was done by Christoph Mandl (1980). He modeled the problem and designed a two-step solution to solve it: first generating a set of initial feasible routes, and then modifying them heuristically. Most of his techniques were tested when developing the operators of our genetic algorithm, and they include: adding a node to a route if the demand that gets covered by doing so is high enough, removing a node if the demand that stops being covered by doing so is low enough, and exchanging parts of routes. We make use of similar operators when generating initial solutions for our genetic algorithm, as will be described in Chapter 4.

In the work of Mandl, a well-known benchmark was published: Mandl's Swiss road network. It can be seen in Figure 1.1. Many authors have used this benchmark to test their solutions, thus producing a good amount of data and results that become very useful when evaluating new solutions to the UTRP. As an aside, our graphical representation of the network is not the same as the original one, even though every edge has the correct length. That happens because we do not have the distances between every node, and this is a possible configuration of the network, even if not planar. The edge between n_3 and n_{11} crosses two nodes. This may be imagined as a bridge that crosses over two bus stops on a lower road, and is not physically connected to them. Throughout this work, we use this representation for Mandl's network.

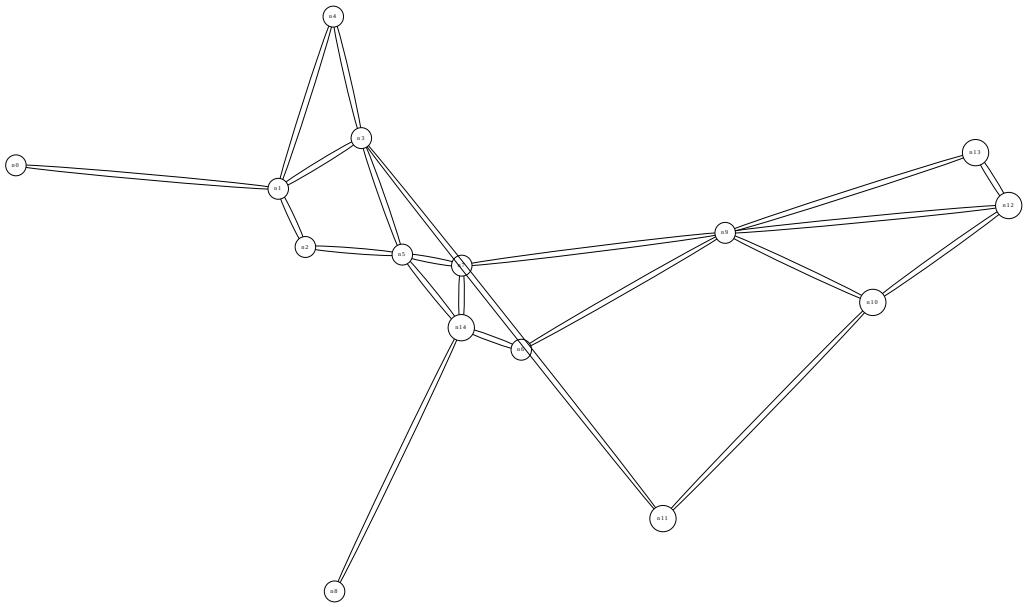

Figure 1.1: Mandl's Swiss road network.

As an example for real world directed solutions, Israeli and Ceder (1989) created

a complex and interactive multi-step approach, where some steps were to be computed with exact methods, and others decided by the traffic planner. The approach was not fully implemented, having been tested only for the first steps. Even though a practical solution to the problem is the actual goal when solving the UTNDP, using a more abstract version of the problem facilitates research and allows better comparison of intrinsics, such as algorithms and techniques. This leads to better knowledge about the problem, which will then lead to better real word solutions.

K. Wan *et al.* (2003) proposed a Mixed Integer Formulation for Multi-Route Transit Network Design. It attempts to attend all demand while minimizing total operator costs. While resembling our formulation, they have a more complex definition that handles route frequencies, and their model is non-linear, also because of the more detailed definition of the problem. Therefore, their formulation gets *linearized*. It is also not multi-objective, only considering total operator costs. If using our problem definition, assuming every node has demand, and only considering total operator costs, the solutions to this hypothetical single-objective version would be trivial and directly derived from the minimum spanning tree of the network, as will be discussed in Chapter 4.

In the exact approach of Borndörfer *et al.* (2007), a multi-commodity flow model is proposed. It generates line routes dynamically, much like our approach, but it assumes that a *precomputed* small set of possible routes is available, in order to decrease the search space. Further, it considers the maximization of the demand covered without transfers as a main goal, but thus ignoring transfer waiting times in optimization. It can be employed to test bigger instances, and was tested with data of Potsdam, a medium-sized city in Germany.

Considering a problem definition equivalent to the one used here, Chakroborty *et al.* (2002; 2004) use a genetic algorithm and propose a new *fitness* function that measures how good the genetic pool of potential solutions is. It uses the absolute difference between the path a passenger takes and the shortest possible path (if there were routes everywhere). Then it scales this value according to the demand covered by the path. It also takes unsatisfied demand into consideration.

The algorithm was tested with Mandl's Swiss road network, and it achieved better results than what was known at that time, showing the potential of genetic algorithms in solving the UTRP. This work used a very different approach from ours for initial route set generation and different genetic algorithm operators, and it did not treat the multi-objectiveness of the UTRP, optimizing only the user satisfaction.

When experimenting and comparing results with the work of Chakroborty *et al.*, we found some inconsistencies in the published quality factors for their obtained route sets. This will be discussed in more detail in Chapter 5.

When generating and selecting route sets, the user travel times, considering transfers, is the most often used decision factor. In contrast with that, in the genetic algorithm of Tom (2003) *et al.*, a bigger set of criteria is considered, such as operator costs, highest directly covered demand and fleet sizes (given their problem definition included frequencies for each route). It is worthy to note that our initial solution generating procedure also selects routes using different criteria, such as operator costs and covered demand. This leads to higher quality and differentiation in initial route sets.

Agrawal *et al.* (2004) focus on parallelizing genetic algorithms for the UTNDP. They use a large benchmark, with 1332 nodes and 4076 edges to test their techniques. They achieved a performance improvement of about half times the number of processors, i.e. with $2x$ processors their experiments ran x times faster than with one processor. Our metaheuristic approach is also theoretically parallelizable and could take advantage of speed-ups in order to process larger networks, although a coordination system would have to be implemented in order to distribute work among processors. The largest benchmark tested in our work has 110 nodes and 275 links.

A genetic algorithm was also proposed by Wei Fan *et al.* (2006). Their problem definition is more complex, dividing nodes into centroid and non-centroid, and considering multiple transport modes (e.g. bus and train), each with different and dynamic demands. Their solution is based on creating an initial set of feasible solutions, given restrictions on route size, based on the shortest paths of the network. Then, the genetic algorithm is used to select and optimize a best route set out of the initial set. The operators of the genetic algorithm are the same as in previous works. They shortly discuss the redesign of existing networks as well, tackling the problem mainly by fixing some of the existing routes and only optimizing the others.

Wei Fan *et al.* (2008) also used Tabu search to solve the more specialized version of the UTRP discussed in the previous paragraph. They propose three different Tabu search algorithms and compare them with their previous genetic algorithm. Sensibility analysis is also performed, after which numerical results are obtained. The algorithm is, however, also based on initially generating all feasible routes incorporating problem constraints. This would not be (generally) applicable using our problem definition since there would be too many feasible routes. Furthermore, the used benchmarks are not openly available, which prevents direct comparison of techniques and results.

As an example of a relatively new ad-hoc graphical tool developed and used for a specific city is presented in the work of Álvarez *et al.* (2010). They created an interactive and attractive tool to design routes and position bus stops automatically. They divide their data, for the city of Burgos, Spain, in four types, one for each type of day, e.g. school day and holiday. This corresponds to using four different Origin-Destination demand matrices. The approach fixes the number of routes, and the initial and final destination of each route. The process of generating routes is based on inserting nodes within existing routes, and applying local search operators, such as operators that exchange parts of routes between each other. It also takes turns optimizing bus schedules and routing. But, since it is an ad-hoc solution and it has not been tested on common benchmark instances, its techniques cannot be directly compared to our algorithms.

An interesting approach to the UTRP, differing from most, is that of Curtin *et al.* (2011). Instead of minimizing the cost for user or operator, they maximize the service level of each edge. This level attempts to take into consideration more factors, such as the importance of the route for political and geographical reasons, and combines user and operator costs as constraints to the problem, rather than as objectives. The problem is solved using mixed integer programming, and optimizes one route at a time, thus not calculating the interplay of multiple routes in service quality. The authors also assume that more potential stops along an edge equates to more potential served demand, even though demand coverage relates more to where the route leads to than to the number of covered stops. To solve the problem formulation in reasonable time, certain restrictions

on route size are taken advantage of in order to limit the number of nodes that must be considered simultaneously. Such a divide-and-conquer approach is useful to allow mathematical formulations to be applied to bigger networks.

A recent article by Mazloumi (2012) makes a comparison between the ant colony and genetic algorithm metaheuristics in the context of scheduling (UTSP). Here one is interested in an optimal timetable, avoiding too full or empty transport vehicles. The metaheuristics are compared in terms of efficiency and accuracy to provide optimal solutions. Both had similar accuracy, finding near optimal solutions, but ant colony achieved the solution in less iterations, showing it has a more intelligent search approach. Even though interesting, this is more strongly connected to the actual problem being treated (UTSP) and how the operators and evaluation functions were defined than to intrinsic differences between the metaheuristics. It has therefore no direct implications when considering ant colony and genetic algorithms for the UTRP.

The most closely related work to ours has been by Lang Fan *et al.* (FAN, 2009; FAN; MUMFORD; EVANS, 2009; FAN; MUMFORD, 2010). Their main idea is to define the problem more generically to allow comparison of techniques and algorithms rather than problem statements and restrictions.

It is, of course, important to characterize the problem completely and exactly in order to achieve good solutions for real world scenarios. But, when developing and testing new algorithms, the definition should remain the same, and as general as possible, so that it does not stand in the way of evaluating these techniques.

Lang Fan *et al.* focus on that, and we enforce it, following the same definition they use, with exception of making it more general in regard to constraints on the number of routes and on cycle allowance. We also employ the *mutator* operator of the genetic algorithm of Lang Fan *et al.* Finally, we use this and previous work, based on Mandl's network, to test and compare our techniques

2 PROBLEM STATEMENT

2.1 Urban Transit Network Design Problem

The Urban Transit Network Design Problem consists in finding and defining adequate vehicle routes for an urban network that satisfies user demand over the region while fulfilling practical constraints. The user demands include: being able to get to destinations fast, with the least number of transfers, not having to wait much for the transport and avoiding crowded vehicles.

It is important to notice that the first two items are mostly affected by where the transportation goes through, that is, the routes used. The last two aspects are more related to the frequency and scheduling of vehicles in a route. When considering practical constraints, one normally has to deal with limits in the number of routes and their lengths, possibility to cross certain regions, and the costs of creating and maintaining the public transport system. Other constraints are due to stake-holders, such as government, businesses and taxpayers. Citing Lang Fan (2009), while many parties will benefit from an efficient public transport service, each one will evaluate its service from their own perspective.

As can be seen from the above description, two main parts of the UTNDP may be pointed out: the routing of vehicles and their scheduling. This classification is widely used, and the name of these parts are: the Urban Transit Routing Problem and the Urban Transit Scheduling Problem. Often, solutions to the UTNDP tackle each of these two aspects separately.

Changing transit routes is often more costly, as it may involve actually constructing the route (e.g. with railroads), and because the population living next to the network, including public transport users, drivers and others, may be affected. Therefore planning is a very important and decisive step. On the other hand, scheduling is adapted throughout the day, and, even though there are also costs involved in changing scheduling, they are of lesser magnitude when compared to routing change expenses.

Therefore, we attempt to focus mainly on the UTRP, considering it the step that demands highest attention currently. In the next two sections, the UTSP and UTRP are described in more detail, and then we present the formal definition used for the UTRP throughout our work.

2.1.1 Urban Transit Scheduling Problem

Given a set of routes and a number of available vehicles, deciding the exact times and frequencies at which vehicles will pass through every access point and thus make routes is the concern of the UTSP. The effects of scheduling are mainly the passenger's waiting times, the crowdedness and the costs in fulfilling defined timetables, including fuel, energy, personnel and maintenance associated.

Waiting times may be further divided, since the initial waiting time, i.e. prior to beginning the public transport part of the trip, can be avoided if a user heads to the access point at the right time and schedules are maintained. The waiting time outside of a vehicle and before getting to the destination is the transfer time and is often unavoidable. The last waiting time, sometimes neglected, but also important, is caused by the time distance between different vehicles in a route. Even if the schedules are maintained and the user plans ahead, it may still have to wait at his destination after arriving, because every public transport option is either too late or too early.

Mixed Integer Programming solutions are commonly proposed for scheduling, mostly with non-linear objective functions. Nevertheless, since it is also a hard problem with an unfeasibly high number of possibilities, scheduling is often targeted by heuristics and metaheuristics. These algorithms have to deal with constraints such as: limited fleet size, limited vehicle capacity, stopping time bounds (in an access point), interdependence between stopping times in consecutive access points, nondeterministic travel times due to traffic and nonuniform arrival of passengers (CHAKROBORTY, 2004).

2.1.2 Urban Transit Routing Problem

The urban transit routing problem consists of finding a set of traffic routes, given passengers, operator and further constraints, that achieves good average travel times, low number of transfers, low costs for the operator, or a combination of these goals. Since scheduling is unknown in this step, a frequency value is often associated with each route, or the frequencies are considered equal. Here we assume equal frequency per route to maintain simplicity and compatibility with previous works. Passenger cost is proportional to the time in order to fulfill the demand represented by the OD matrix, while operator cost is often associated to the length of routes.

To develop algorithms for it, we must precisely define the UTRP. We choose a more general definition in order to allow a comparison with previous work and focus on techniques and algorithms rather than problem fidelity. The next definition follows the work of Lang Fan *et al.* (2009).

2.2 UTRP Definition

A graph $G(V, E)$ represents the *transport network*, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes representing predefined bus stops, train stops, or more broadly, access points where the transport is able do pick up and drop off passengers, and where $E = \{e_1, \ldots, e_n\} \subseteq V \times V$ represents the set of direct physical connections between nodes.

A *route* in the *transport network* $G(V, E)$ is a path $r_a = (v_{i_1}, \ldots, v_{i_q})$, where $i_k \in \{1, \ldots, n\}$. A set of routes $R = \{r_a : 1 \leq a \leq N\}$, where N is the number of possible routes, defines a solution to the UTRP problem.

2.2.1 Objective Function

To evaluate the quality of a route set, one must first define a *route network*, which contains only the edges of the respective route. We define as E_a the set of edges of the route r_a. Then, the *transit graph* can be defined as a graph $H(V', E')$ in which $V' \subseteq R \times V$. The node $w_{ia} \in V'$ in the *transit graph* is a pair that combines the route r_a and the node $v_i \in V$ from the *transport network*. Consequently, we define the edges in E' in two parts. E'_1 corresponds to the set of edges within individual routes, and E'_2 represents transfers:

$$E'_1 = \bigcup_{r_a \in R} \{(w_{ia}, w_{ja}) : (v_i, v_j) \in E_a\}, \quad E'_2 = \bigcup_{v_i \in V} \{(w_{ia}, w_{ib}) : v_i \in r_a \cap r_b\}.$$

Given the *transit graph*, one can define two cost functions, one measuring the user cost and one the operator cost. The operator objective function is defined as:

$$C_O(R) = \sum_{r_a \in R} \sum_{e \in E_a} c(e),$$

where $c(e)$ is the cost of operating edge e.

To define the user cost, we use t_e, the time required to travel through edge e, and the travel penalty t_{pen}, which is the time it takes to make a transfer. The value t_{pen} is assumed to be constant since we do not deal with scheduling. The value t_{pen} also includes a time penalty regarding the inconvenience of having to make a transfer instead of staying on the same route. Given that, the edges in E'_2 have length t_{pen}, while the edges $(w_{ia}, w_{ja}) \in E'_1$ have length $t_{(v_i, v_j)}$. Now, the minimum journey time in the *transit graph* TG from v_i to v_j, $\alpha_{ij}(TG)$, is defined as the shortest path from a node in $\{w_{ia} : v_i \in r_a\}$ to a node in $\{w_{jb} : v_j \in r_b\}$.

Let d_{ij} denote the transit demand from node v_i to node v_j (defined as the number of passengers wishing to travel from v_i to v_j). Assuming the passengers will always choose to travel on the shortest paths, the user cost functions TTT (Total Travel Time) and ATT (Average Travel Time) can be defined as follows:

$$TTT = \sum_{(v_i, v_j) \in V \times V} d_{ij} \alpha_{ij}(R), \qquad ATT = \frac{TTT}{\sum_{(v_i, v_j) \in V \times V} d_{ij}}$$

Since there is more than one objective function to be optimized in this problem, solutions can be classified as *dominated* or *undominated*. In a set of solution candidates, a solution s is *undominated* if and only if no other candidate in the set is better than s on both quality measurements.

2.2.2 Restrictions

Two main restrictions are the only applied in our definition: every route must be a path, and a route set must cover the whole demand (without limits on transfers), i.e. not leave any passengers unattended or unable to reach their destinations. In variants of the problem, further restrictions are often enforced. It is common not to allow cycles or backtracks within routes, or to limit the size of routes, and it is often assumed that routes are undirected, i.e., that the public transport travels in both directions of a route.

Nevertheless, it is not uncommon there to be different paths depending on the direction of the route, and cycles are known to occur. Therefore, and in order to obtain greater generality, we do not assume any of those restrictions as necessary, and all of our algorithms are able to deal with any subset of desired restrictions, within those mentioned above, and produce solutions that satisfy them. This facilitates comparison with algorithms for a broader class of problem definitions.

Another common characteristic in previous models of the UTRP is fixing the number of desired routes. This may or may not be realistic, since on the one hand having more routes may result in more costs, but on the other hand, one already considers operator costs within the operator cost function. Thus, we choose to allow setting of a lower and upper bound on the number of routes. We do this since implementing our solutions for a variable number of routes is not a burden, and it allows more generality and thus broader comparison possibilities.

2.2.3 Bounds on the Number of Routes and on the Objective Functions

Given a *transport network*, one can extract minimum bounds on both objective functions. These can be used to measure the quality of achieved results when other experiments have not been performed on the same network, or just to calculate the maximum possible optimality gap.

To obtain the best possible value for the passenger travel time, it is enough to assume that every passenger will travel through the best path in the *transport network* (as opposed to the *transit graph*). This clearly gives the minimum obtainable travel time since every value in the OD matrix is multiplied by the lowest number possible. To calculate minimum paths in a graph, Dijkstra's or Floyd-Warshall's algorithm (DIJKSTRA, 1959; FLOYD, 1962) can be utilized.

To minimize operator cost, one must find the route set with the least cost that allows demand to be fulfilled, i.e. that connects every node on the network. This is solved by calculating the minimum spanning tree of the *transport network* (KRUSKAL, 1956). By definition, a minimum spanning tree is the subset of edges that connects the graph with the least weight. This is exactly the same structure needed to minimize operator costs: a subset of edges that connects the graph with the least cost. A solution with the same cost as the minimum spanning tree is the solution where every edge in the minimum spanning tree is a route. Also, many of these routes may be combined in order to obtain a smaller number of routes, as is discussed in Section 4.3.

An exception has to be made in the case that not every node possesses demand. In this case, connecting the demand nodes with the least weight corresponds to finding a

minimum Steiner tree (HWANG; RICHARDS, 1992), and not a minimum spanning tree. Nevertheless, given that the edge lengths in the *transport network* are usually metric, the network satisfies the triangle inequality. Therefore, the minimum spanning tree for the nodes with demands costs no more than double the cost of the optimal Steiner tree (VAZIRANI, 2001), and is thus a reasonable approximation for the optimal solution.

Finally, we also attempt to obtain upper bounds on the number of nodes needed to fulfill all demand without any transfers. This time, the approach is not exact, and is based on a greedy algorithm, Algorithm 1. It create routes that maximize instant demand coverage gain (without transfers). The final route set's size can be used as an estimator of the number of routes to be used on the network. The algorithm tends to create routes that are as long as possible, since they cover more demand, until no more neighbors that enhance demand coverage are found, or if the route length limit is reached. An example for a route set created using this algorithm on Mandl's network is shown in Figure 2.1.

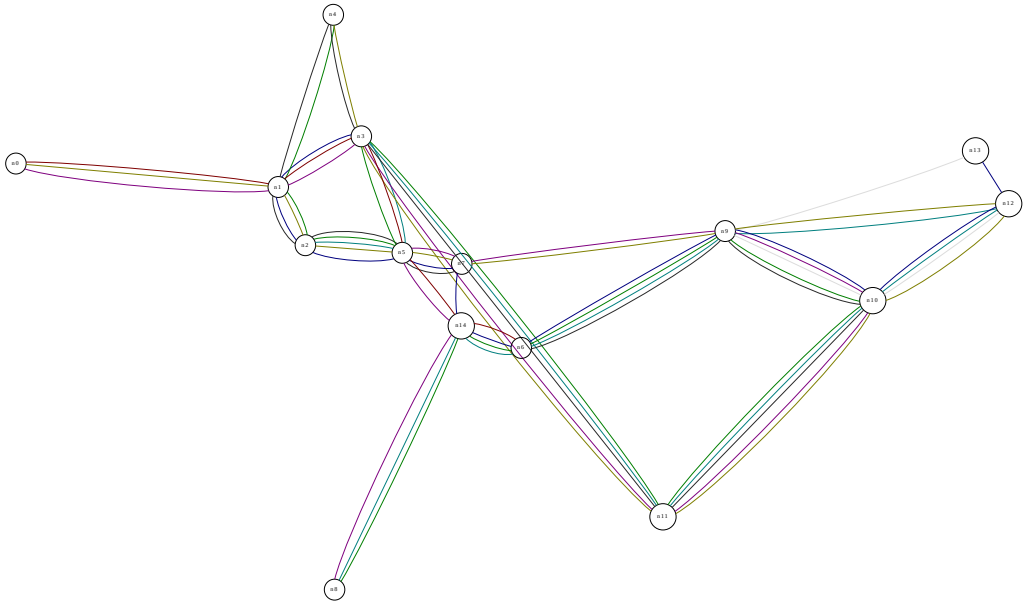

Figure 2.1: Route set obtained by applying Algorithm 1 to Mandl's Swiss road network.

Algorithm 1: Greedy demand coverage.

Data: network $G = (V, E)$, demand matrix $d_{ij} \in \mathbb{Q}^{V \times V}$, covered matrix $c_{ij} \in \mathbb{B}^{V \times V}$ and $routeSetSize$

Result: route set RS with high directly covered demand

for $i, j \in V$ **do** $c_{ij} \longleftarrow 0$

`// ` c_{ij} `is` 1 `iff demand between i and j`
`// is satisfied without transfers`

repeat
- Route $r \longleftarrow \emptyset$
- $(i^*, j^*) = argmax_{(i,j) \in X^2 | c_{ij}=1} d_{ij}$
- **if** $d_{i^*j^*} = 0$ **then break**
- fill r with a path between i^* and j^* using DFS
- **repeat**
  - **for** $i, j \in r$ **do** $c_{ij} \longleftarrow 1$
  - $maxDemand \longleftarrow 0$
  - **for** $i \in V$ **do**
    - **if** i *can be added to* r **then**
      - $gainedDemand \longleftarrow 0$
      - **for** $j \in r$ **do** $gainedDemand \longleftarrow gainedDemand + d_{ij}(1 - c_{ij})$
      - **if** $maxDemand < gainedDemand$ **then**
        - $maxDemand \longleftarrow gainedDemand$
        - $toAdd = i$
      - **endif**
    - **endif**
  - **endfor**
  - **if** $maxDemand > 0$ **then** add $toAdd$ to r
- **until** $maxDemand \leq 0$
- add r to RS

until $|RS| \geq routeSetSize$

3 MIXED INTEGER PROGRAMING FORMULATION

In this section we present a formulation that fully solves the UTRP using mixed integer programming.

3.1 Variables

Our formulation uses the following decision variables: Let R be a set of routes. For every route $r \in R$, s_{rv} indicates if vertex v is the initial vertex of route r. Similarly, f_{rv} indicates if v is the final vertex of route r. The variable x_{re} is 1 if and only if the edge $e \in E$ belongs to route r.

For every pair of vertices v and w, s^*_{vwr} indicates if the best path between v and w starts on route r. Likewise, f^*_{vwr} indicates if r is the final route of the best path between v and w. Finally, $x^*_{vwe'_{rnsm}}$ indicates if edge $e'_{rnsm} \in E'$ is on the best path between v and w, where E' is the set of edges of the *transit graph*, as explained in Section 2.2.

Additionally, the real variables p_{rv} indicate the position, starting from 0, of the vertex v on route r.

3.2 Constraints

For every route $r \in R$, we have the following constraints:

$$\sum_{v \in V} s_{rv} = 1 \tag{3.1}$$

$$\sum_{v \in V} f_{rv} = 1 \tag{3.2}$$

$$\sum_{e=(v,w) \in E} x_{re} + f_{rv} = \sum_{\bar{e}=(w,v) \in E} x_{r\bar{e}} + s_{rv} \qquad \forall v \in V \tag{3.3}$$

$$\sum_{e=(v,w) \in E} x_{re} + f_{rv} \leq 1 \qquad \forall v \in V \tag{3.4}$$

$$|V| - |V| s_{rv} \leq p_{rv} \qquad \forall v \in V \tag{3.5}$$

$$p_{rw} - |V| + |V| x_{re} \leq p_{rv} + 1 \qquad \forall (v,w) \in E \tag{3.6}$$

$$p_{rw} + |V| - |V| x_{re} \geq p_{rv} + 1 \qquad \forall (v,w) \in E \tag{3.7}$$

Additionally, for every pair of vertices $v, w \in V$, we have the constraints:

$$x^*_{vwe'_{rnrm}} \leq x_{re} + x_{r\bar{e}} \qquad \forall r \in R, e = (n,m) \in E \tag{3.8}$$

$$x^*_{vwe'_{rnsm}} = 0 \qquad \forall n \neq m \in V, r \neq s \in R \tag{3.9}$$

$$\sum_{r \in R} s^*_{vwr} = 1 \tag{3.10}$$

$$\sum_{r \in R} f^*_{vwr} = 1 \tag{3.11}$$

$$\sum_{\substack{s \in R, \\ m \in V}} x^*_{vwe'_{rnsm}} + [w = n] f^*_{vwr} = \sum_{\substack{s \in R, \\ m \in V}} x^*_{vwe'_{smrn}} + [v = n] s^*_{vwr} \qquad \forall n \in V, r \in R \tag{3.12}$$

There are also constraints for improving performance, such as not allowing multiple transfers on the same node and path, not allowing cycles in the best paths, and ordering the routes such that shorter ones have smaller indices. These constraints only decrease search space by removing non-optimal solutions or duplicates of solutions, i.e. solutions that are equal to others except for the route ordering.

3.3 Objective Functions

Given that $c(e)$ is the cost for operating edge e, t_e is the time for traversing edge e, d_{ij} is the demand between nodes i and j, and t_{pen} is the time penalty for making a transfer between routes, the following two objective functions are defined (as explained in Section 2.2.1):

$$\textbf{minimize} \quad \sum_{\substack{r\in R,\\ e\in E}} c(e)x_{re} \qquad \text{(operator cost)}$$

$$\textbf{minimize} \quad \sum_{\substack{r\in R,\\ v,w\in V,\\ e=(n,m)\in E}} t_e d_{vw} x^*_{vwe'_{rnrm}} + \sum_{\substack{r,s\in R,\\ v,w,n\in V}} t_{pen} d_{vw} x^*_{vwe'_{rnsn}} \qquad \text{(user travel time)}$$

3.4 Discussion

The formulation begins by fixing the number of routes $|R|$ (where $R = \{1, 2, \ldots, |R|\}$) in the solution route set. This implies in multiple optimizations when a range of number of routes is acceptable, one for each size.

Each route must then be defined. For each route $r \in R$ there are $2|V| + |E|$ binary variables that define it completely. Constraint (3.1) and (3.2) guarantees that there will be unique start and end nodes for each route. Constraint (3.3) assures that every node within a route will have an equal number of outgoing and incoming edges active in the route, and Constraint (3.4) assures that this number shall be smaller or equal to one, except for the start and end nodes, which shall have zero incoming or outgoing edges, respectively.

To remove closed loops which may be left in the previous steps, the p_{rv} variables and the constraints (3.5), (3.6) and (3.7) are used. Constraint (3.5) sets the position of the first node in a route to 0. Constraint (3.6) and (3.7) set the position of a node that comes after another node in a route to one plus the position of the previous node, assuring sequential positioning. Therefore, the first node will have the smallest position. Since a closed loop has no start, no potential position can successfully be assigned to a node in it. It should be kept in mind that, if extra closed loop routes are acceptable in a solution (i.e. if the number of routes can be bigger than the given $|R|$), these constraints can be removed. These restrictions are similar to the Miller, Tucker, Zemlin subtour elimination constraints in formulations of the traveling salesperson problem and similar problems (MILLER; TUCKER; ZEMLIN, 1960).

The rest of the formulation finds $|V|^2$ shortest paths, for each pair origin-destiny, based on the available routes, following a standard approach for shortest paths LP formulations. Constraint (3.8) assures that best paths will consist only of edges available in the chosen routes. Constraint (3.9) disallows moving between routes, except when this movement is from a node to itself, in which case it would characterize a transfer (and thus is allowed).

Constraint (3.10) and (3.11) guarantee that there will be unique start and end routes for each best path. Constraint (3.12) assures that every best path will actually be a path, by balancing the number of ingoing and outgoing edges of every node (which is, in this case, a pair in $R \times V$, as explained in Section 2.2.1).

The variables s^*_{vwr}, f^*_{vwr} and $x^*_{vwe'}$ are actually implemented as **real** variables between 0 and 1. If their values end up being non-integer on an optimal solution, the solution is still valid and can be interpreted as follows: the best path can be taken in several different manners, and each manner is used in proportion to the value of the corresponding variable. e.g. if s^*_{vwr} is 0.5, then 50% of the demand from v to w starts on route r. Furthermore,

every single path has the same length, since otherwise the rest of the demand would also use the shortest path, and this would provide a solution which is better than the optimal, a contradiction. This also means that there is at least one integer solution which is equivalent to any optimal non-integer solution found.

Finally, using mixed integer programming, one can only optimize a single objective function. To handle this, we use two approaches: either summing both functions, each weighted by an importance factor, or setting one of the goals as a constraint, and not allowing it to be higher than a certain maximum value. This gives the traffic planner the possibility to find the best solution for one of the goals, respecting boundaries on the other, or to find the best solution given a certain *linear trade-off* between the two objective functions.

3.5 Modeling the Redesign of an Existing Network

A common problem when designing new routes for an existing public transport network is to decide whether to change existing routes or not. This may incur various costs such as those of building new routes, changing terminal locations and even getting the population aware of the new possibilites.

It is easy to use our MIP formulation to help solve this problem. First, one can define a cost for adding an edge e to route r, A_{re}. Similarly, the cost to delete edge e from route r is D_{re}. One must also know the previous configuration of the routes on the network: $P \subseteq R \times E$ contains every pair (r, e) containing an edge e that is used in route r on the previous configuration. With this, an objective function for the modification costs can be defined as

$$\textbf{minimize} \quad \sum_{(r,e)\in P} D_{re}(1 - x_{re}) + \sum_{(r,e)\notin P} A_{re}x_{re}. \qquad \text{(modification cost)}$$

As with the two previous objective functions, either a *linear trade-off* must be found between user, operator and modification costs, or they can be set as constraints, limiting costs to a certain threshold while minimizing other objectives.

3.6 Divide-and-Conquer Approach

Since solving the MIP formulation for a large number of nodes is prohibitively slow, we propose a *divide-and-conquer* algorithm that optimizes the network in multiple steps. It is heuristic, and is has some open details regarding which routes should be maintained in each step, and which nodes should be used to connect routes in these same steps. We describe it here to present the idea, not to give a detailed definition.

In each step, groups of nodes are selected in the network, with up to K nodes, where K is a parameter that influences how much time each mixed integer program will run.

These groups must be connected, i.e. there must be a path between every member of the group without going through nodes that are outside of the group. For each group, the best route set using R routes is calculated.

Then, in the end of a step, the distance between the groups of nodes is calculated as an average of the distances between the nodes in each group. If there is no connection between nodes in two groups, then these groups are not connected.

After doing this, the groups of nodes are collapsed into nodes, i.e. one group becomes one node, and the distances and connectedness between these new nodes are the distances and connectedness between the groups of nodes, as described in the last paragraph. With this, a new step begins. This is done until the total number of nodes is less than or equal to K.

To extract the solution from the resulting graph, nodes are expanded back into groups of nodes. The routes between the nodes are now transformed in order to travel through the uncollapsed graph, going into and out of the corresponding groups. Inside each group, the route being transformed is connected to an existing route, chosen randomly. The path used to connect two routes can be the best path between the nodes in the extremities of each route, or another arbitrary path. This is guaranteed to be possible because an edge in the collapsed graph corresponds to connectedness in the uncollapsed graph, and because each group is connected within itself. Some routes might not get selected to form a route from a previous step. Some or all of these unselected routes might be excluded in order to reduce the total number of routes.

This is performed until the original graph is obtained, now with a set of routes, which is the solution of the algorithm. Restrictions regarding cycles in routes, minimum and maximum lengths are not considered.

4 METAHEURISTIC APPROACH

The metaheuristic used here is a genetic algorithm (HOLLAND, 1975). Its main structure is presented in Algorithm 2. Each solution in this implementation corresponds to a set of routes, where each route is represented as a sequence of nodes. To keep track of solutions, we use three vectors. The vector P stores the population, and has a size of $populationSize$ (or less if the population is getting built or rebuilt). The vector U stores every solution that is $undominated$ (following the concept explained in Section 2.2), and is not bounded in size. The vector B is created when generating initial solutions and stores some fundamental route sets related to the *transport network*, as discussed in Section 4.1. It is not modified after the end of the initialization routine.

Analyzing the structure of the algorithm, we see that the first step is the generation of initial routes, or loading the state from a previous run with the same input data. The initialization procedure is explained in Section 4.1. After creating the three route set vectors (B, P, U), the algorithm enters its main loop. In most of the iterations, every element in the population, one at a time, is cloned and suffers a mutation, called *small change*. The *make small change* procedure is from the work of Lang Fan *et al.* (2009), and is explained in Section 4.2. This new route set is compared to every existing route set in the population and, if it dominates any of the existing solutions, it substitutes the first dominated route set found, taking its place in the population.

Once every $applyOperatorsInterval$, the population is emptied. Before that, every undominated solution is saved on U, so that high quality solutions do not get lost. Then, the population P is rebuilt with random route sets from B and U. This happens until the population is half full. After this, existing route sets in the population are matched up and a random operator is applied to them. The operators that are available for choice in this step are discussed in Section 4.4. This generates a new solution, that is then added to the population. The process goes on like this until the population is full, and the algorithm is ready to run further.

4.1 Generation of Initial Solutions

For the generation of initial solutions, the first step is the creation of a base list of solutions, called B here. This base list of route sets contains fundamental solutions that are characteristic to the network, as explained in Section 2.2.3.

Algorithm 2: Genetic algorithm for the UTRP.

Data: network $G = (V, E)$, demand matrix $d_{ij} \in \mathbb{Q}^{V \times V}$, totalGenerations, applyOperatorsInterval, populationSize
Result: a vector of undominated route sets U
vector<route set> B, P, U
if *previous state available* **then**
  $B, P, U \longleftarrow$ load previous state(G,d)
else
  $B, P, U \longleftarrow$ generate initial solutions(G,d)
endif
$iteration \longleftarrow 0$
repeat
  $iteration \longleftarrow iteration + 1$
  if $iteration \bmod applyOperatorsInterval = 0$ **then**
    $U \longleftarrow$ update undominated solutions(P)
    $P \longleftarrow \emptyset$
    $i \longleftarrow 0$
    while $i < populationSize/2$ **do**
      $i \longleftarrow i + 1$
      $rs \longleftarrow$ select random route set(B,U)
      add rs to P
    endw
    while $i < populationSize$ **do**
      $i \longleftarrow i + 1$
      $a, b \longleftarrow$ choose two random route sets(P)
      $oper \longleftarrow$ choose a random operator
      $rs \longleftarrow oper(a, b)$
      add rs to P
    endw
  endif
  for $rs \in P$ **do**
    $rs' \longleftarrow$ make small change(rs)
    for $p \in P$ **do**
      if rs' *dominates* p **then**
        substitute p with rs' in P
        break
      endif
    endfor
  endfor
until $iteration \geq totalGenerations$
$U \longleftarrow$ update undominated solutions(P)
save state(B,U)

One of these is the minimum spanning tree of the *transport network*. The second source of routes is the subgraph of the *transport network* containing only shortest paths given by the Dijkstra or Floyd-Warshall algorithm. All routes contained in one of those two subgraphs are joined into a route set, and the route sets get simplified (as described in Section 4.3).

Another source of viable routes for B is the solution obtained by Algorithm 1, as explained in Section 2.2.3. These route sets tend to cover a high amount of demand directly, avoiding transfer penalties. Nevertheless, they are usually quite longer than shortest paths, so they are not very useful if transfer penalties are low.

Considering that, we developed a second greedy algorithm that tries to maximize demand cover, but now using shortest paths. Algorithm 3 is the result of that idea, and its solution is also added to B. The route set provided by this algorithm is often of high quality for the passenger.

Finally, the last source of viable solutions for B is the relaxed solution of the MIP formulation. This is discussed in Section 4.1.1.

Then, the extraction of valid route sets begins. The extraction respects minimum and maximum number of nodes in route, existence of cycles and number of routes per route set. It is a random procedure, possibly extracting a different part of the route or route set at each time it is executed. An example of an invalid route in a scenario that does not allow cycles is shown in Figure 4.1, as well as the extracted route, with cycles removed.

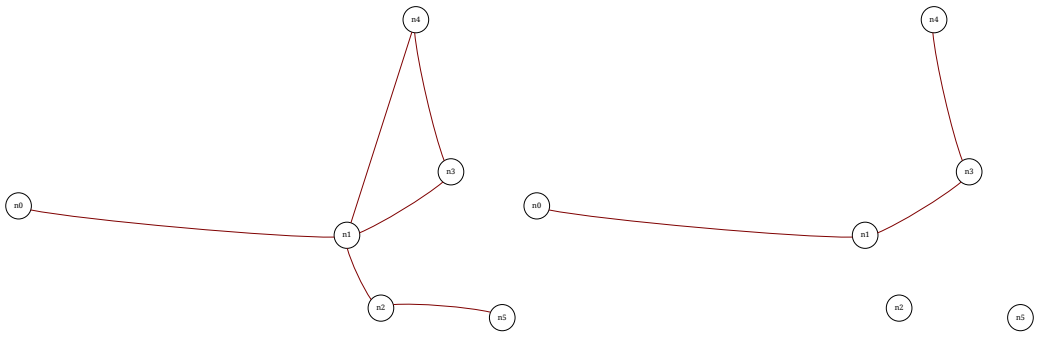

Figure 4.1: Route containing cycles before and after extraction.

The extractions are then performed several times for each route set (since each extraction may have a different outcome), and all of the resulting valid solutions initialize the base list. Then, a variable number of exchanges, explained in Section 4.4.1, is executed between random valid solutions. At the end, every route set gets simplified, and the base list is ready. Meanwhile, the undominated list U is also fed with every undominated solution in the base list B.

The next step is actually creating the initial population. This is done by taking one third of $|P|$ solutions from the base list, one third of $|P|$ solutions from the undominated list, and making exchanges among members of P until the remaining third is filled.

Algorithm 3: Greedy demand coverage using best paths.

Data: network $G = (V, E)$, demand matrix $d_{ij} \in \mathbb{Q}^{V \times V}$, covered matrix $c_{ij} \in \mathbb{B}^{V \times V}$ and $routeSetSize$
Result: route set RS with high directly covered demand using best paths
for $i, j \in V$ **do** $c_{ij} \longleftarrow 0$
`// ` c_{ij} `is 1 iff demand between i and j`
`// is satisfied without transfers`
repeat
  Route $r \longleftarrow \emptyset$
  $maxDemand \longleftarrow 0$
  for $i, j \in V$ **do**
    $gainedDemand \longleftarrow 0$
    for $a, b \in$ `BestPath(`i, j`)` **do**
      $gainedDemand \longleftarrow gainedDemand + d_{ab}(1 - c_{ab})$
    endfor
    if $maxDemand < gainedDemand$ **then**
      $maxDemand \longleftarrow gainedDemand$
      $bestPair \longleftarrow (i, j)$
    endif
  endfor
  if $maxDemand = 0$ **then**
    break
  endif
  $r \longleftarrow$ `BestPath(`$bestPair$`)`
  add r to RS
  `Simplify(`RS`)`
  update coverage matrix c
until $|RS| \geq routeSetSize$
repeat
  $changed \longleftarrow false$
  for $rs \in RS$ **do**
    update coverage matrix c
    for $i, j \in V$ **do**
      if $c_{ij} = 0$ *and (*r *does not contain* i*) and (r contains j)* **then**
        if *one of the extremities of* r *is neighbor of* i **then**
          add i to r in the corresponding extremity
          $changed \longleftarrow true$
        endif
      endif
    endfor
  endfor
until *!changed*

4.1.1 Route Suggestions from MIP Relaxed Solution

The linear relaxation of our MIP formulation can be found in reasonable time. It works by relaxing the constraints of every binary variable (x_{re}, s_{rv}, f_{rv}), allowing them to assume real values between 0 and 1. Instead of obtaining routes in the solution, we obtain sets of edges with a real value, interpreted as the likelihood of the edge being used within the route. From this point on, there are two approaches that we follow: considering each route separately, as described in Section 4.1.1.1, or joining them, as explained in Section 4.1.1.2.

4.1.1.1 Separate routes

Here, we choose a route from the MIP formulation, and begin on one of its start nodes, randomly, with chance proportional to s_{rv}. Then, we traverse edges in a similar way, but with chance proportional to x_{re}. Along the way, we keep the *minimum probability*, p_min, which is the minimum value between s_{rv} and the used x_{re} variables. When we reach an end node, the route is complete, unless p_min is higher than f_{rv} for this node, in which case we keep traversing edges with probability equal to $p_min - f_{rv}/p_min$, and adjust $p_min \longleftarrow p_min - f_{rv}$.

Route suggestions from this approach are actually routes that can be used by the passengers in the (interpretation of the) MIP formulation, even if not by every passenger, since the demand from one node to another corresponds to a flow of 1 that travels between them, and only up to the *minimum probability* flows through the extracted route.

An example of an extracted route is shown in Figure 4.2. For the chosen route, there were only one start and one end node with non-zero probabilities. The start node is blue, whereas the end node is red. Other nodes are black. The values for the variables x_{re} for the chosen route are in the edges labels: the first value corresponds to the probability of traversing the edge in the direction given by the edge, and the second value corresponds to traversing it in the opposite direction. In this scenario, loops were not allowed, and routes could be traversed in both directions (the given one and the opposite direction). This means that swapping the red and blue colors and inverting all edges would lead to the same quality factors.

4.1.1.2 Global importance

Here, we sum, for each edge, all the likelihoods that it receives in every route, and this value is interpreted as the importance of the edge, i.e. $importance_e = \sum_{r \in R} x_{re}$. For the start ($s_{rv}$) and end ($f_{rv}$) nodes, similarly, $startImportance_v = \sum_{r \in R} s_{rv}$ and $endImportance_v = \sum_{r \in R} f_{rv}$.

To create routes based on the importance value of each edge, we simply perform a random walk, choosing start node or end node with probability proportional to the importances of each node, and then traversing edges in a similar manner (proportionally to their importances). We also need to define a desired length for the route, or another stopping criterion. We choose here to end within a certain length range, and each value of the range has the same probability of being picked as the desired route length.

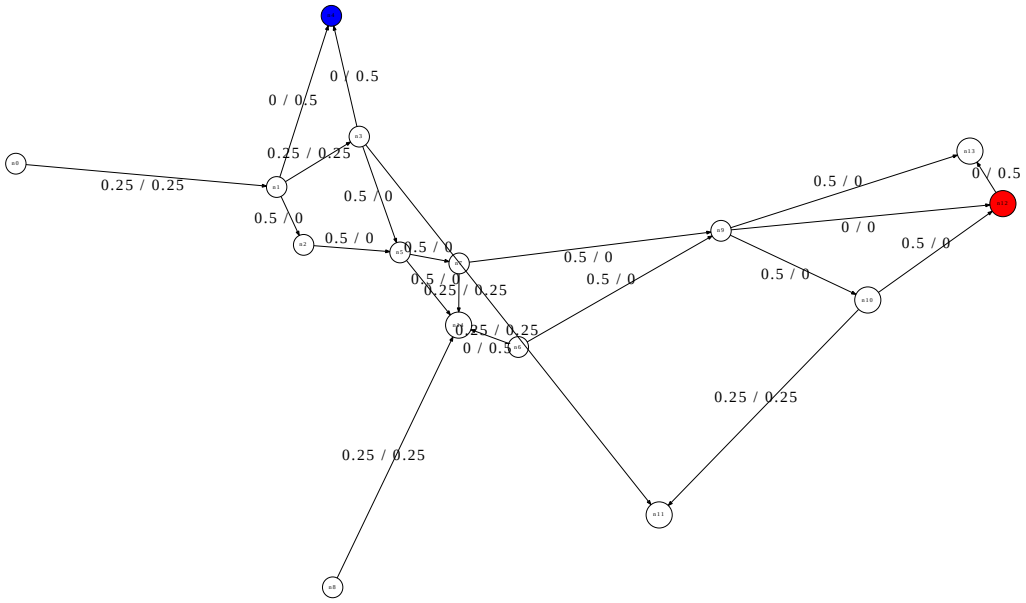

Figure 4.2: Route suggestion network for one route.

An example of an extracted route using this approach is shown in Figure 4.3. We can see that there are three different importance levels: 0, 0.5 and 1, values which can be interpreted as low, medium and high importances. In this example, the number of routes was 2, the operator cost limit was not applied, and the number of edges used, as is explained in Section 4.1.1.3, was set to 15, with a resulting ATT of 10.62 minutes.

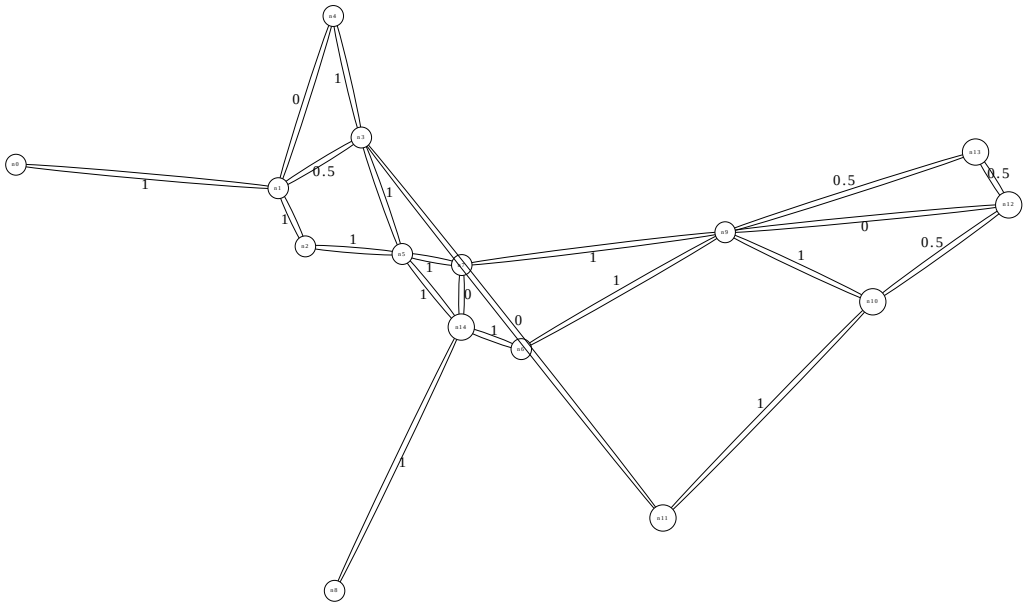

Figure 4.3: Route suggestion network with global importances.

4.1.1.3 Parameter tuning

The first important parameter that should be configured in order to obtain useful suggestions from the MIP relaxation is the sum of total edges chosen, i.e. $\sum_{r \in R, e \in E} x_{re}$, which is related to the operator cost, but is not the same since in this case the edge lengths

are not considered.

If this value is too high, the best achievable ATT could be obtained with less edges. This would mean that the extra edges in the solution are irrelevant, and can be chosen randomly, not affecting the ATT, although changing the importance values. In other words, the effect would be of random noise on the importance values. On the other hand, if this value is too low, then the ATT will be too high, and we will thus obtain a route suggestion from a scenario with travel times that are higher than desired, rendering the suggestions less useful.

Besides, the number of routes and operator cost should also be set appropriately. Each route in the relaxed solution may contain several traversable routes within it, but they still need to be balanced according to the constraints of Section 3.2. To allow a higher number of routes to be generated, the number of routes should be increased. Similarly, the operator cost limits the number of edges used, but considering edge lengths. Configuring it is similar to configuring the sum of total edges chosen, as described in the previous paragraph.

4.2 Mutation

The mutation operator is based on the *Make-Small-Change* procedure, from the work of Lang Fan *et al.* (2009). It applies very small changes to routes, and thus navigates through the neighborhood of solutions. This is important in order to find local minima, maximizing the potential of some route set, but must be complemented so that other areas of the solution space may also be explored.

The only mutations that are executed are the addition and removal of nodes at the start or end of a route. Only one node is added or removed at a time. As a last step, the simplification operator, explained in Section 4.3, is applied. This guarantees that route sets do not have unnecessary routes and thus obtain better passenger and operator costs.

4.3 Route Set Simplification

When finding a shortest path between two nodes, one will travel exactly through the shortest paths between the intermediate nodes. This is a known characteristic of problems to which dynamic programming can be applied, and here it is taken advantage of in a different way.

As explained in Section 4.1, some route sets are determined and saved on a list of base route sets. This list will be used to build new initial route sets and candidate solutions. One of the source of routes for the base route sets is the shortest path between nodes. These are all added into the routes sets of the base list. But, as discussed above, there are many overlaps between these shortest routes.

Besides, when using a random decision based algorithm, changes to routes can naturally lead to the same situation: some routes can be *contained* in others. Even when not so, two routes may still be *joinable* without any disadvantage to users or to operators. This is formally defined as follows: the route r can be *joined* with s if: $\exists_{0 \leq i \leq |s|-1}$:

$\forall_{1 \leq j \leq min(|r|,|s|-i)} : r_j = s_{j+i}$. Two routes r and s are joinable if either one of them can be joined with the other.

These ideas lead to the development of an operation that attempts to combine every pair of routes belonging to a route set. This operation leads to big savings of operator costs for the initial route list, and improves the route set during the execution of the genetic algorithm. An example of a simplification being applied is shown on Figure 4.4.

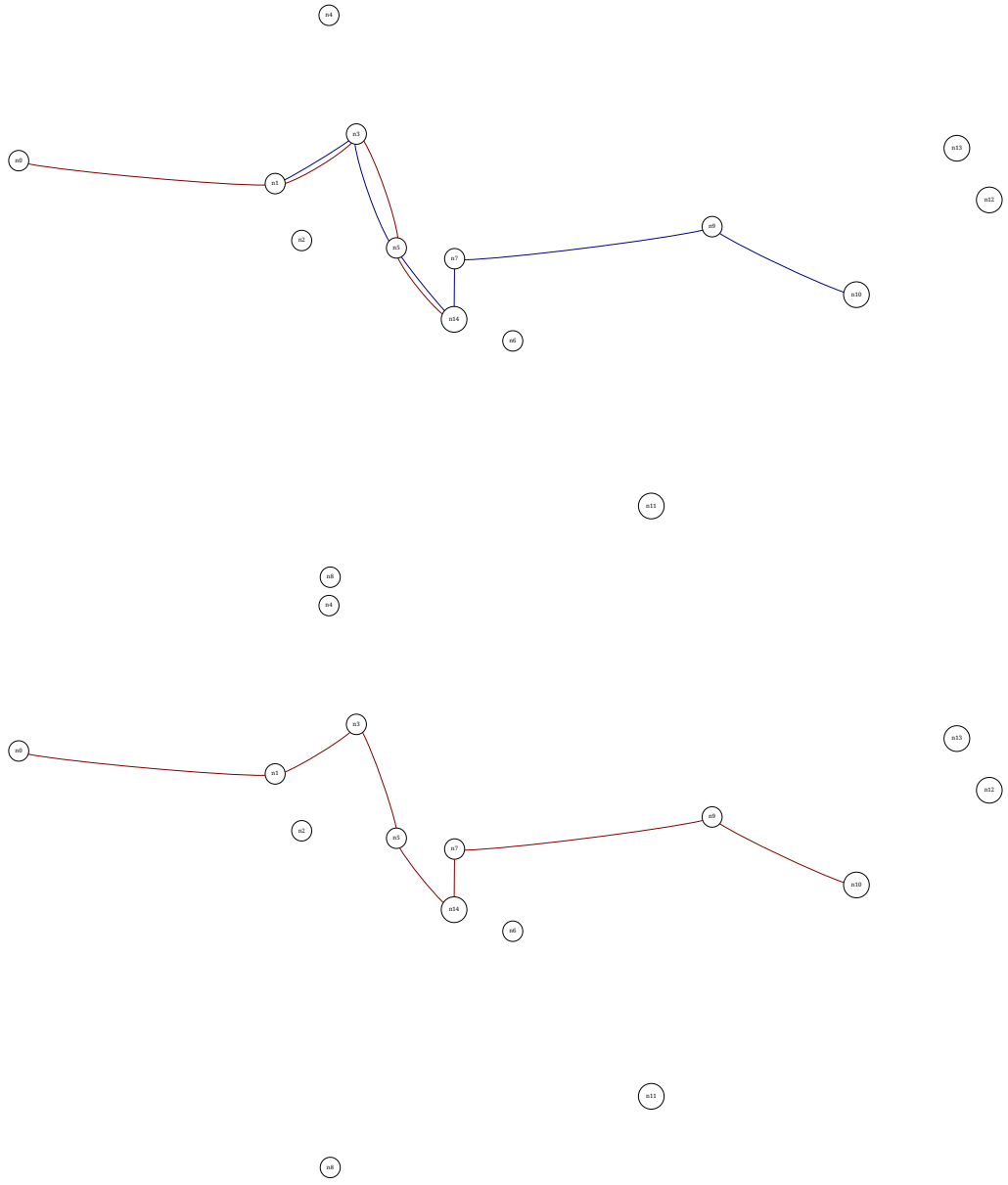

Figure 4.4: Route set before and after simplification.

4.4 Operators

4.4.1 Exchange

The route sets consist of many routes. This operator simply exchanges some of the routes, chosen randomly, between route sets, and simplifies the route sets afterwards. This is useful to explore new solutions, but does not actually *modify* routes. An example of this operation can be seen in Figure 4.5.

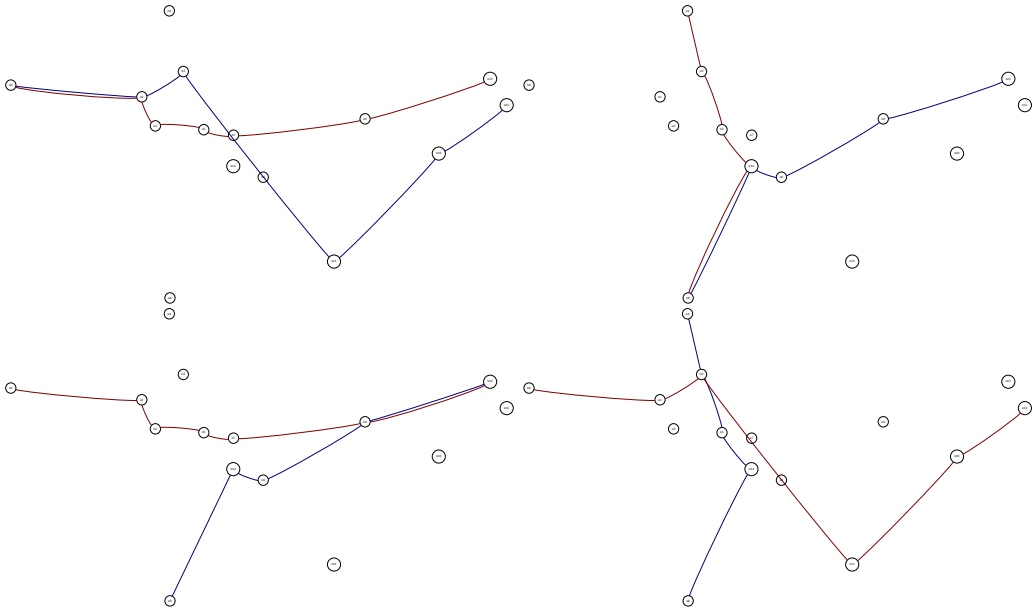

Figure 4.5: A pair of route sets before an exchange (above) and after it (below).

4.4.2 Crossover

In order to try to join two routes and perhaps keep qualities of both of them, a crossover operation was also defined. The operation works as follows: a cut point is randomly defined in both routes, based on a shared node between the routes. From the beginning of the first route until its cut point, all nodes are copied into the result. From there on, the nodes are copied from the cut point of the second route until its end. After that, all the currently applying restrictions are considered, such as prohibition of cycles and limits on the number of nodes on a route. Nodes are excluded from or included into the resulting route until it satisfies all restrictions.

In Figure 4.6, two routes are shown together with the resulting route, after a crossover.

4.5 General Considerations

As discussed in Section 1.1, genetic algorithms have shown to be successful in solving the UTRP. In this section we discuss what are the main characteristics and differences of our approach in comparison to previous ones.

The first different aspect of our approach is the representation of routes and route sets,

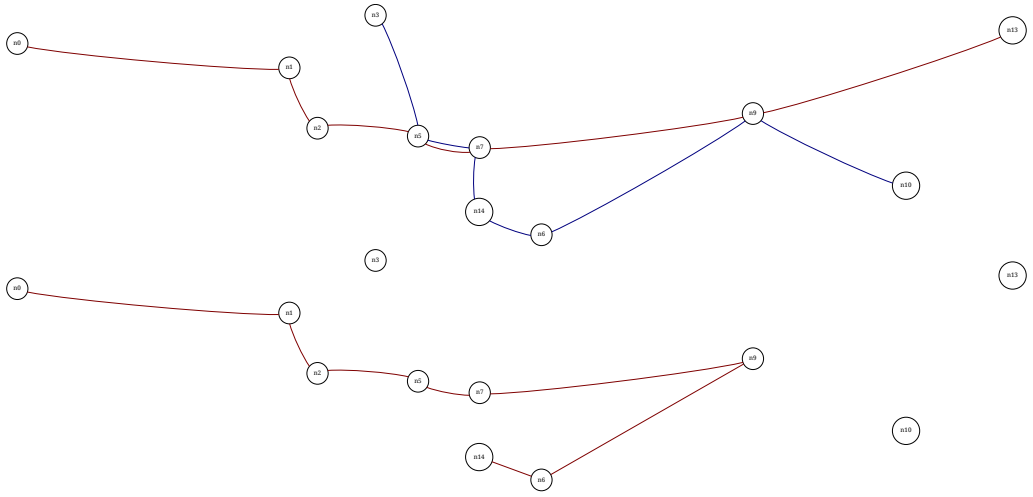

Figure 4.6: Routes before and after a crossover ($n9$ is the cut point).

that is more complex as commonly seen (AGRAWAL; MATHEW, 2004; CHAKROBORTY, 2004; FAN; MUMFORD; EVANS, 2009). It consists of dynamic data structures such as sets and vectors instead of simple strings. This is justified because representing, moving, copying and modifying routes is not the bottleneck in this problem, but the *evaluation* of route sets. This means the structure is made more generic, easier to program and to adapt to new restrictions, without significant performance loss.

The next difference are the mutation and the operators. The mutation, which applies very small changes to routes, navigating through the neighborhood of solutions, functions like a local search, and is executed much more frequently than operators. This is important in order to find local minima, maximizing the potential of some route set. Nevertheless, local search must be complemented so that other areas of the solution space may also be explored, and that is where the operators of exchange and crossover come in. This tends to better explore the solution space, or at least avoid convergence in local minima.

The key aspect of our approach is the set of different sources used for generating good initial solutions. Besides using the shortest paths, as has been done very often before, we use the MIP relaxation, two greedy algorithms and the Minimum Spanning Tree (as discussed in Sections 2.2.3 and 4.1).

The last important characteristic is the maintenance of a base and an undominated solution list, apart from the population. The first one is used for the initialization procedure, and the second one maintains every undominated solution found. Every time a new solution is created by an operator, it may not dominate any solutions, but still be undominated. Since the population size is constant, this new undominated solution would be lost. Instead, it is kept in the undominated list, whose size is dynamic.

Since this is a multi-objective optimization problem, we only consider *domination* for the classification of solutions. Thus, there is no fitness level that rates and orders solutions, and all are considered equal as long as one does not dominate another. Therefore, when applying operators, every solution candidate has the same probability of being chosen. Besides that, every element in the population suffers exactly one mutation per evolution step.

4.6 Performance Analysis

To assess the performance of the developed genetic algorithm, we analyzed the complexity of it, and performed profiling.

Regarding complexity analysis, the evaluation of a route set, the most expensive step of the algorithm, is done using Dijkstra's algorithm. In this way, a solution can be evaluated in time $O(|R|^2|V|^2 \log(|R||V|))$, since there are $O(|V|)$ evaluations, one per node, and each one corresponds to an execution of Dijkstra's algorithm over a graph with $O(|R||V|)$ nodes and $O(|R|^2|V|)$ edges.

The overall running time of the algorithm would be, then, for a population of size P and T evolution steps, $O(PT|R|^2|V|^2 \log(|R||V|))$.

Then, to experimentally assess the performance of our implementation, we used the open source GNU Profiler. The graphical output can be seen in Figure 4.7. As shown, more than 90% of the execution time is spent running Dijkstra. This confirms that evaluation is the bottleneck for this algorithm. When analyzing the profiling results, one should keep in mind that functions with less than 0.5% activity are removed, and, besides that, since the program is compiled with the most agressive optimizations possible, many functions are joined with others by the compiler and disappear from the results. An actual calling tree for the program would be, thus, much more complex.

Since the evaluation of different route sets may be done simultaneously, the exploration of the search space may be accelerated arbitrarily, as long as increasing P, through parallelization. This would require, though, an extension to the current implementation to coordinate many concurrent executions of the algorithm by receiving the current undominated solutions, and generating new populations to be run on different processors or machines. This has not been implemented yet, and will be discussed further in Chapter 6.

Another possibility that is considered in pursue of increasing performance is to only allow addition of nodes to routes during mutation. This would change the algorithm a bit, since removal of nodes would have to be done separately, after a bigger number of iterations, but could prove effective. When adding a node to a route, the *transit graph* gets only slightly changed.

After such an addition, a single $O(|R|^2|V|^2)$ step (assuming $|V| \geq \log(|R||V|)$ that updates every distance is enough. This step consists of updating the distance from the new node to every other one (by using Dijkstra's algorithm), and then trying to use the node that was just added between every pair of nodes (similar to one step of Floyd-Warshall's algorithm), updating the distances if appropriate. This step could also make use of a graphical processing unit (GPU), at least when updating the distance between every pair, since a step of Floyd-Warshall's algorithm is parallelizable and can make full use of GPU processing capabilities (KATZ; KIDER JR, 2008). This has also not been implemented yet, and is discussed in Chapter 6.

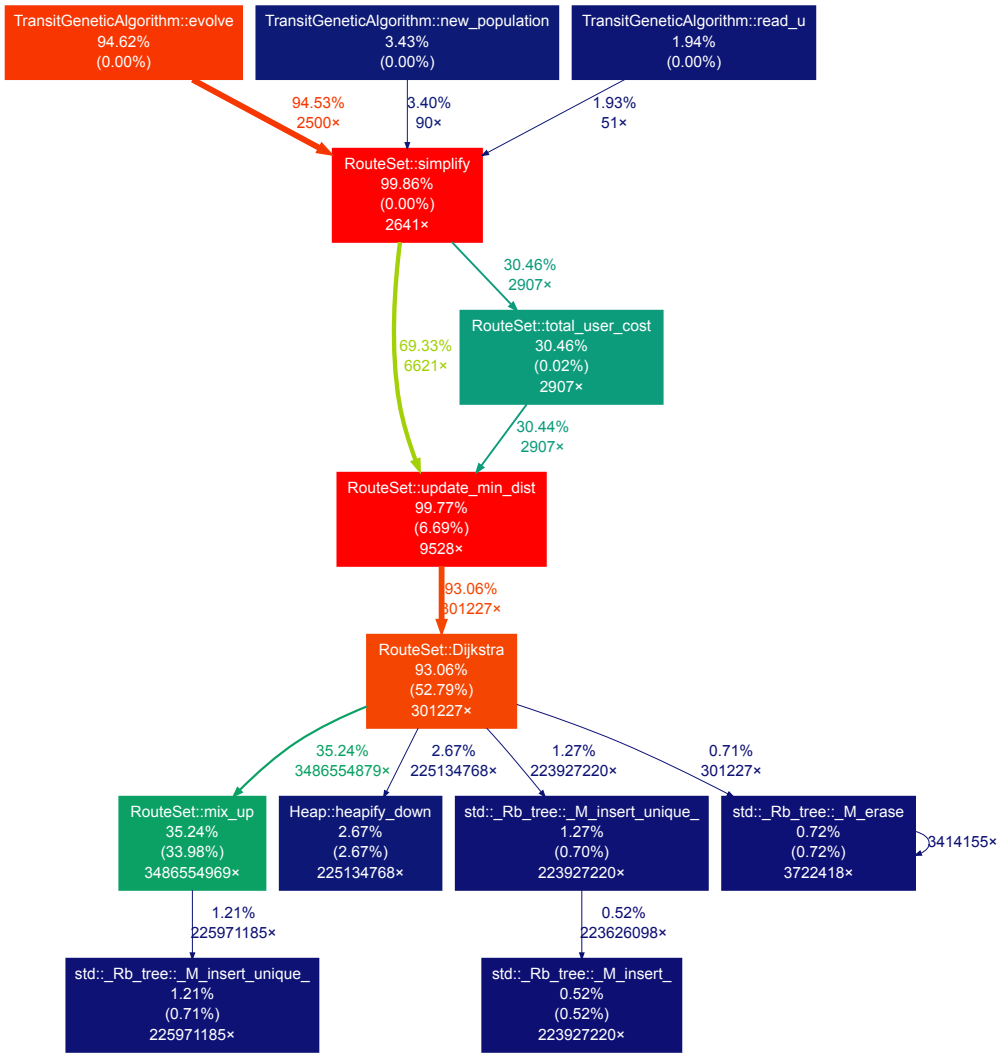

Figure 4.7: Profiling results for the implementation of the genetic algorithm.

5 EXPERIMENTAL RESULTS

As discussed in Section 1.1, there is one commonly available benchmark for the UTRP, namely Mandl's Swiss road network (MANDL, 1980), which has 15 nodes and 21 links. We use it for our experiments in order to compare our results with previous works (BAAJ; MAHMASSANI, 1991; SHIH; MAHMASSANI, 1994; KIDWAI, 1998; CHAKROBORTY, 2004; ZHAO, 2006; FAN; MUMFORD; EVANS, 2009).

We have used both our MIP formulation, as described in Section 5.1, and the metaheuristic, as discussed in Section 5.2.1, with Mandl's network, also obtaining exact answers for small numbers of routes.

Besides using the well known benchmark, we choose to test our algorithm with a medium sized network as well. This network has also been tested before by the authors whose results are the best known for Mandl's network. This larger test case was produced by Lang Fan *et al.* (2009), and by using it we expect test our solution's quality and scalability for bigger networks.

All results are evaluated using the following quantities, as in previous works:

- d_i is the percentage of the demand satisfied with i transfers.
- ATT is the average travel time (in minutes per passenger), including transfer penalties.
- C_O is the cost for the operator, i.e., the total route length (in minutes, considering constant transport speed).

Among these, the only quantity that should be as high as possible is d_0, which indicates how much of the demand can be satisfied directly, i.e. without transfers. *All* other measures should be as small as possible.

When evaluating scenarios on which no tests have been previously made, we use the lower bound on the ATT, as explained in Section 2.2.3, to obtain reference quality values.

Finally, results were obtained on a PC with an Intel® Core™2 i5-460M 2.53GHz (3MB L3 cache) processor and 4GB of RAM using Linux Ubuntu 12.04 OS.

5.1 Mixed Integer Programming Approach

The MIP formulation was implemented in GNU MathProg Modeling Language, and solved using the *IBM ILOG CPLEX Optimizer 12.4* (CPLEX, 2009). The execution times have been measured in seconds of real time.

We obtained the best possible route sets on Mandl's network regarding user travel time for two and three routes, which were, to the best of our knowledge, never published before. The quality, processing times and the actual routes of the solutions are given in Table 5.1. Figures 5.1 and 5.2 show the best route sets, for better visualization, for two and three routes, respectively.

Table 5.1: Best possible route sets found using the Mixed Integer formulation

Number of routes	2	3
d_0	84.90 %	93.67 %
d_1	14.00 %	5.43 %
d_2	1.10 %	0.90 %
ATT	11.33 min.	10.50 min.
C_O	98 min.	150 min.
Processing time (s)	1065	78992
Two Routes	6-14-7-5-2-1-4-3-11-10-9-13-12	
	0-1-3-5-7-9-6-14-8	
Three Routes	4-3-11-10-12-13-9-7-5-2-1-0	
	4-3-1-2-5-14-6-9-10-11	
	0-1-4-3-5-7-9-6-14-8	

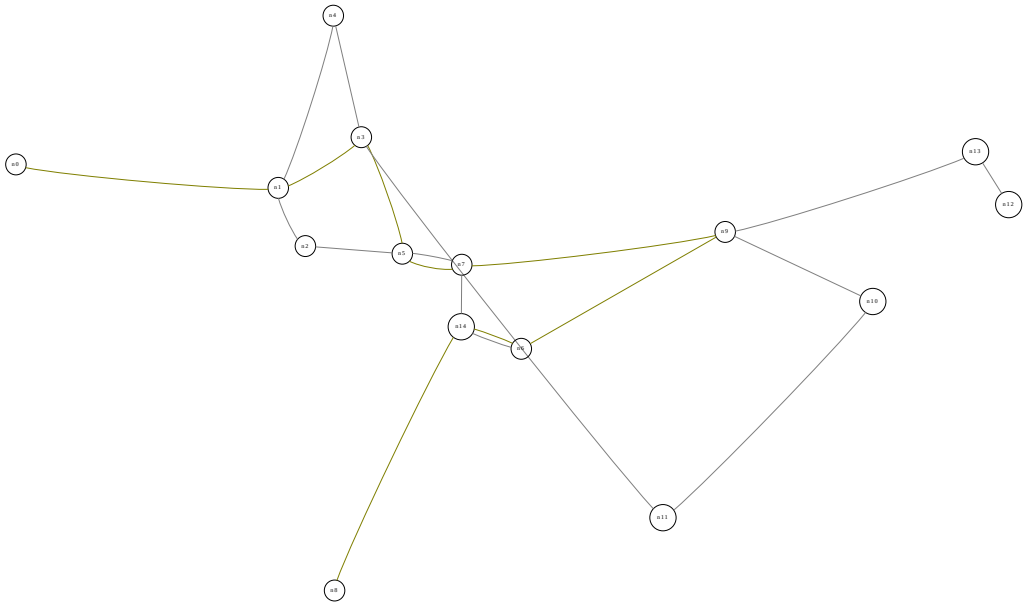

Figure 5.1: Global best route set for two routes.

It is clear that completely solving the UTRP, a **NP-hard** problem, using exact methods is not scalable nor feasible even for relatively small instances of the problem. These experiments show the problem size to which it can still be applied and validate the correctness of the formulation. Besides, other applications for the MIP formulation can take

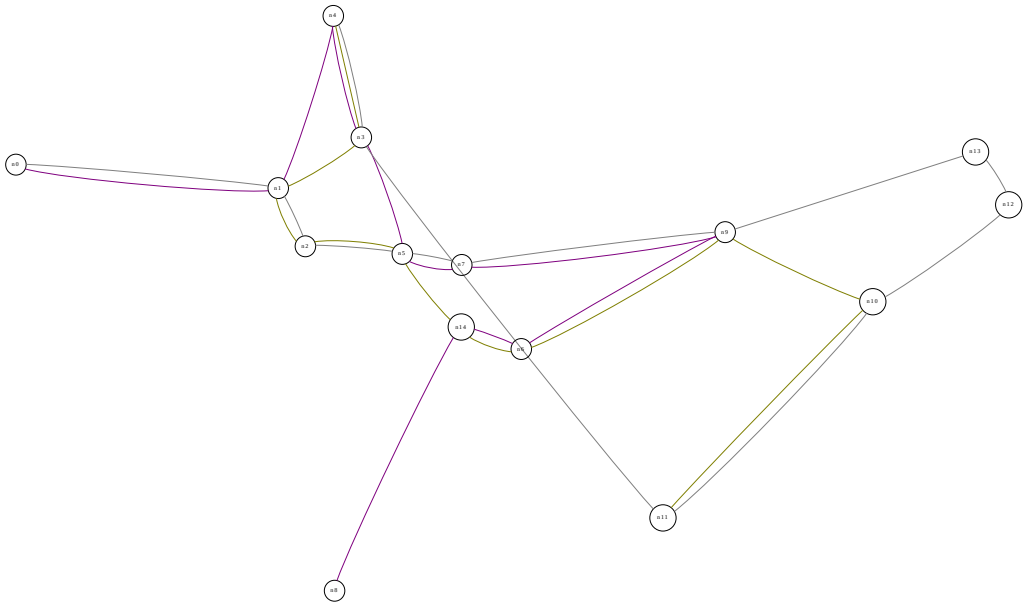

Figure 5.2: Global best route set for three routes.

full advantage of it without bumping into processing time restrictions, such as optimizing a small part of a network, performing many small optimizations on a network in a *divide-and-conquer* fashion, that end up optimizing the whole network (even though not in an exact manner, but heuristic), and finally for suggesting routes by obtaining a relaxed (i.e. real) solution.

Since the 2 routes case has 144 binary variables and the 3 routes scenario has 216, and given the processing times given in Table 5.1, one can estimate the processing time required for solving the instance with 4 routes to be about 77 days. These results were achieved using all available cores (the processing times can be lowered by further parallelization).

5.2 Metaheuristic approach

5.2.1 Mandl's Network

The stopping criterion of the genetic algorithm when applied to Mandl's network was 200, 400 and 600 evolution steps when testing with limits of respectively 4, 6 and 8 routes. The population consisted of 1000 route sets. We chose these values proportional to the ones used in (FAN; MUMFORD; EVANS, 2009), in order to allow a fair comparison, since the actual processing times were not available. Nevertheless, we decreased the number of iterations and total reruns in order to compensate for hardware advances and possible longer execution times per iteration in our approach.

We first compare our results with the multi-objective approach proposed by Fan *et al.* (2009), which is better suited for the UTRP since there are two major concerns when developing routes for urban transit: the quality for the passengers and the costs for the operators. The results are present in Table 5.2. We can see that, in comparison to previously published results, our solutions were always better, i.e., we achieve superior solutions

with equal or smaller prices and better travel times. Moreover, they dominate the previous results, being better in all considered measures, or in only of them, but without affecting the others. The four route sets whose results are shown in Table 5.2 are listed in detail in Table 5.3. Besides, Figure 5.3 shows the graphs, for improved visualization, in the same order as in Table 5.3.

Table 5.2: Comparison between best UTRP multi-objective solutions on Mandl's Network

Scenario	Q_p	Melhor valor conhecido (Lang Fan)	Nossos resultados
Melhor para passageiro	d_0	94.54 %	98.84 %
	d_1	5.46 %	1.16 %
	d_2	0.00 %	0.00 %
	ATT	10.36 min.	**10.10** min.
	C_O	283 min.	**259** min.
Solução balanceada ($C_O \leq 148$)	d_0	93.19 %	93.61 %
	d_1	6.23 %	6.20 %
	d_2	0.58 %	0.19 %
	ATT	10.46 min.	**10.43** min.
	C_O	148 min.	**147** min.
Solução balanceada ($C_O \leq 126$)	d_0	90.88 %	91.23 %
	d_1	8.35 %	7.84 %
	d_2	0.77 %	0.93 %
	ATT	10.65 min.	**10.59** min.
	C_O	126 min.	126 min.
Melhor para operador	d_0	66.09 %	77.78 %
	d_1	30.38 %	21.32 %
	d_2	3.53 %	0.90 %
	ATT	13.34 min.	**12.97** min.
	C_O	63 min.	63 min.

Table 5.3: Route sets found by our metaheuristic for the UTRP on Mandl's Network

Best route set for passengers	**Compromise route set with $C_O \leq 148$**
0-1-3-11-10-12-13-9-6-14-7-5	0-1-2-5-7-9-10-11-3-4
0-1-2-5-14-6-9-10-12-13	4-3-5-7-9-10-12-13
0-1-4-3-5-7-9-10-12-13	6-14-7-5-2-1-3-4
2-5-3-11-10-9-6-14-8	0-1-3-5-14-6
0-1-2-5-7-9-10-12-13	13-9-6-14-8
4-1-2-5-14-8	
4-3-5-14-6	

Compromise route set with $C_O \leq 126$	**Best route set for the operator**
0-1-2-5-7-9-10-11-3	13-12-10-9-6-14-7-5-2-1-0
4-3-5-7-9-10-12-13	4-3-1
6-14-7-5-2-1-3-4	10-11
13-9-6-14-8	14-8
5-14-6	

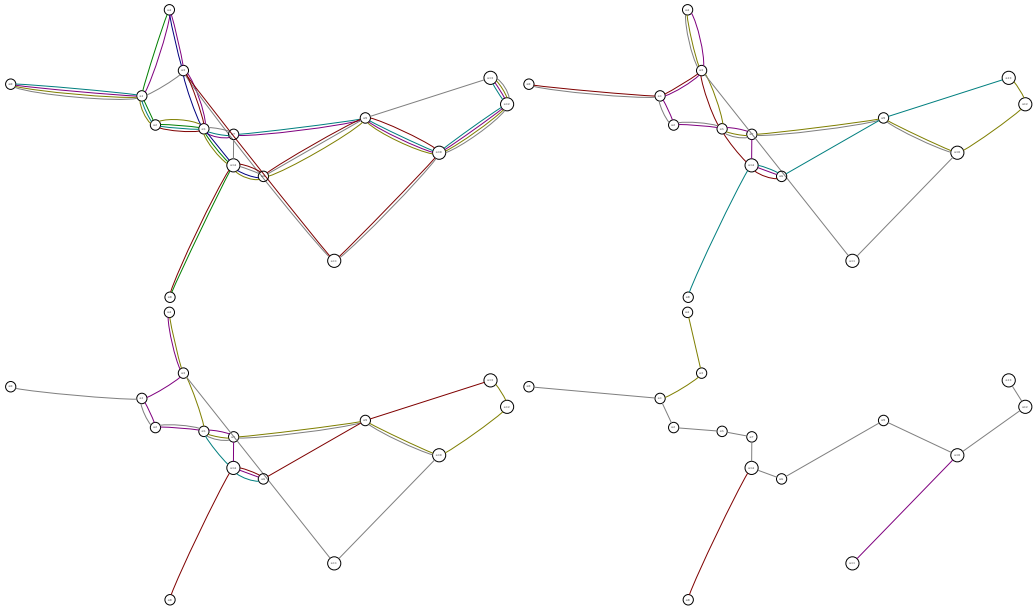

Figure 5.3: Route sets corresponding to Table 5.3.

Finally, to perform a broader comparison, the results on many previous works on Mandl's network (MANDL, 1980); (BAAJ; MAHMASSANI, 1991); (KIDWAI, 1998; CHAKROBORTY, 2004) were analyzed, and the best results are shown in Table 5.4, in comparison to our results. These best values were adapted according to what is described in Section 5.3.

Here, the objective is only one: decrease passenger travel time, without taking transfer penalties into consideration. This was necessary to allow a comparison between heterogeneous penalty values. Not considering penalties favors results that were achieved using lower penalty values, given that these are closer to not having a penalty at all. Since we used five minutes penalty per transfer, and this is the highest amount applied in the cited publications, our results should not be favored.

The average travel time without penalty corresponds to how much time passengers would travel if the penalty was reduced from its current value, t_{pen}, to zero. To calculate this term, the following formula is used (where T_{MAX} is the maximum number of transfers): $ATT_{wop} = ATT - \sum_{i \leq T_{MAX}} t_{pen} d_i i$.

Table 5.4: Comparison between best single-objective UTRP solutions on Mandl's Network

$\|R\|$	Best known ATT_{wop} (CHAKROBORTY, 2004)	ATT_{wop} obtained by our approach
4	10.33 min.	**10.30** min.
6	10.43 min.	**10.11** min.
7	10.53 min.	**10.04** min.
8	11.22 min.	**10.05** min.

It is important to keep in mind that the comparison made in this single-objective case is not as fair as in the multi-objective scenario. This is so because, when comparing results from various sources, slight differences in the definition of the problems occur,

e.g. regarding the minimum and maximum number of nodes per route, allowance of cycles and others. Besides this, the differences in publishing dates, and therefore also in hardware used for running experiments, make it harder to compare the running time of the approaches. Nevertheless, our approach showed to be successful in the single-objective case as well.

As a final remark for this test case, a possible utility of our metaheuristic, besides obtaining route sets, is estimating the curve where the *undominated* solutions lay. This information can be very useful for a planner, since he can better decide on making a *trade-off* between the costs involved. To demonstrate that, we show a graphic in Figure 5.4 with an approximation for the Pareto-optimal curve, i.e. the curve of *undominated* solutions. There, a roughly adjusted function dictates what is the best C_O given a certain ATT, and vice-versa. One can also compare the quality achieved with different algorithm configurations using such a set of Pareto-optimal solutions.

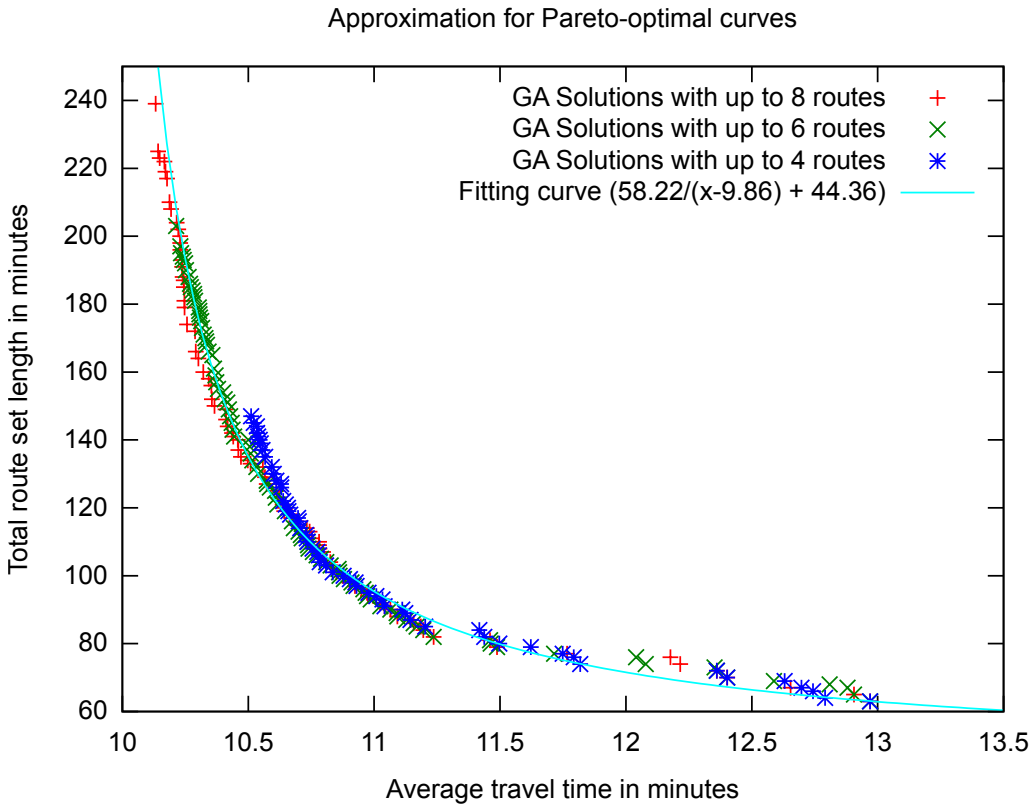

Figure 5.4: Pareto-optimal curves for Mandl's network.

5.2.2 British City Based Network

To assess the behavior of our metaheuristic approach when dealing with larger networks, we applied it to the network defined and used in the work of Lang Fan *et al.* (2009; 2010), which has 110 nodes and 275 links, with 3603360 journeys per day. The Figure 5.5 is a visualization of the network which respects link sizes, but not planarity. The network's size and connectivity are based on a major British city. Two scenarios were defined on top of this network, each one with its own minimum and maximum number of nodes per route, and total number of routes.

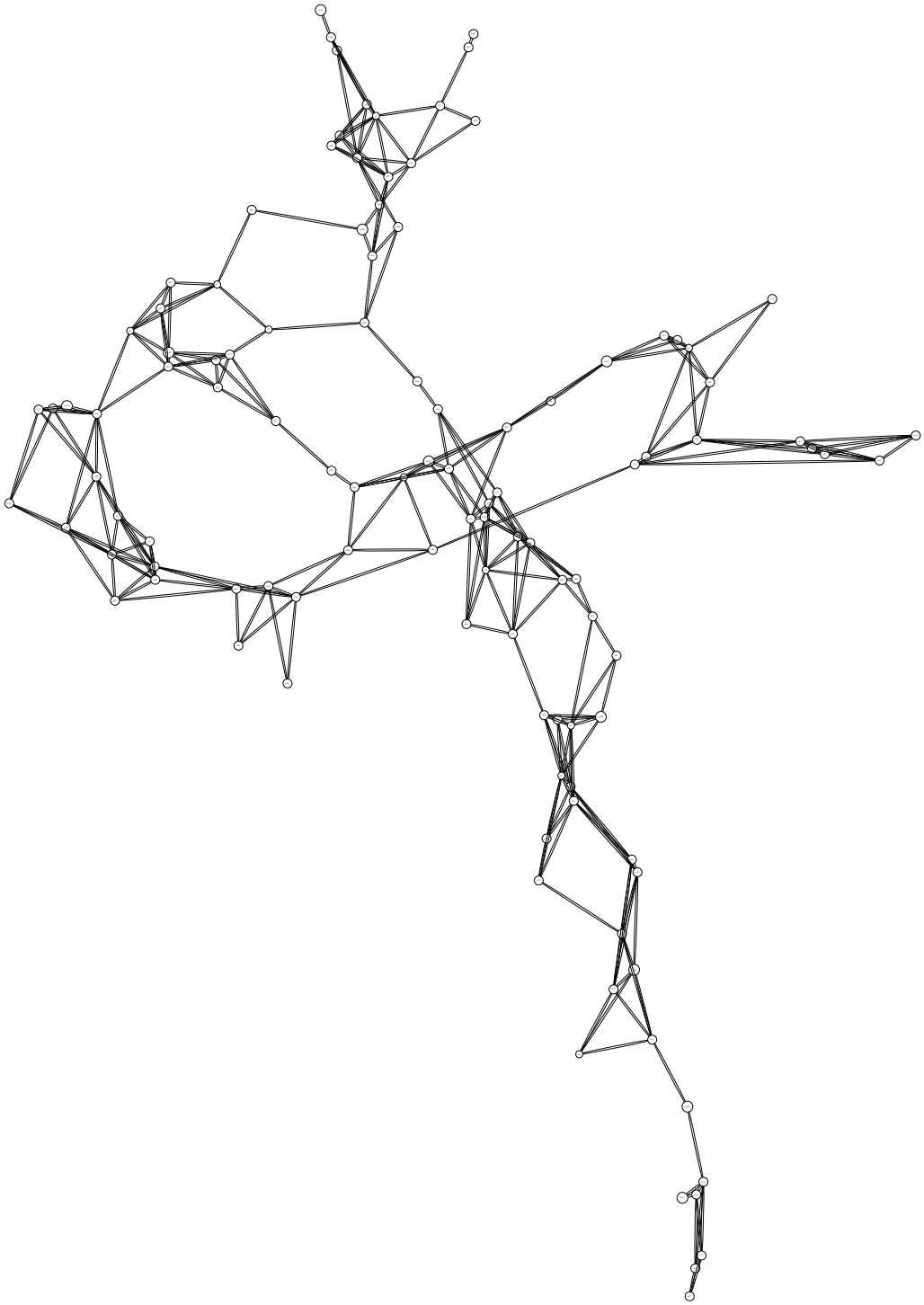

Figure 5.5: British city based network.

Since our approach can handle multiple route set sizes at the same time, the biggest difference in each scenario is the number of nodes per route. The minimum number is 2 and the maximum number is 29 in scenario I, being these limits derived from the *transport network*, its connectivity and number of nodes, as is stated in the work of Lang Fan *et al.* (2009). In scenario II, the minimum number is 10 and the maximum number is 22. The limits in scenario II are derived from the actual routes used in the major British city upon which the artificial network was based.

The running time informed in previous works averages between 13000 and 19000 seconds of processing, depending on the scenario. We used no more than 2500 seconds in each test case.

The full comparison between previous results and our outcome for this network is given in Table 5.5. Two of our route sets on Table 5.5 are shown in Figures 5.6 and 5.7. An intermediate route set found by our metaheuristic, which is undominated and more balanced between passenger and operator cost, is shown in Figure 5.8.

It can be noticed that we optimize the major objective of all scenarios better.

The fact that our route sets were superior when not considering penalties shows that most passengers travel through the fastest or almost fastest paths in the *transport network*. A reason for this is the frequent use of these fastest paths as candidate routes in our algorithm.

Since we also utilize routes created using Algorithm 1 and 3, we are also able to obtain routes with high directly covered demand (i.e. high d_0). If configured not to make transfers, we obtained an ATT of about 57 minutes and only 45 routes. This shows that only 45 routes are needed to cover the whole demand directly, but that this also makes the average trip 20 minutes slower, an increase of more than 50%.

Another important remark is that our outcome had much cheaper (to operate) route sets in the operator-oriented scenarios. This shows that our metaheuristic approach was also successful when optimizing prices instead of travel times. This helps producing a much wider range of available compromise solutions to be chosen by a public transit network planner, thus improving overall network quality.

An interesting fact about the network is obtained by analyzing the output of Algorithm 1 with this network as input. Among the solutions provided by the mentioned algorithm is one that has 45 routes and is able to cover all demand without transfers. Nevertheless, it achieves an ATT of 61 minutes (this is obtained by setting the penalty value very high, such as 5000 minutes, for example). By allowing transfers, the ATT falls to 40 minutes. This shows that 45 routes are enough to cover demand without transfers, and that direct routes can often be inferior to paths with transfers.

5.3 Reproducibility and Difficulties with Previous Results

In this section, we discuss problems faced when comparing our results to previous ones.

By chronological order, the first work used for comparison is that of Chakroborty

Table 5.5: Comparison between best UTRP multi-objective solutions on artificial British city

Scenario	Q_p	Best known value (FAN; MUMFORD; EVANS, 2009)	Our metaheuristic approach results
I-Passenger	d_0	72.91 %	61.95 %
	d_1	20.56 %	37.75 %
	d_2	6.54 %	0.30 %
	ATT	36.28 min.	**36.01** min.
	ATT_{wop}	34.60 min.	**34.09** min.
	C_O	2986 min.	8405 min.
II-Passenger	d_0	71.21 %	53.22 %
	d_1	20.71 %	44.92 %
	d_2	8.08 %	1.85 %
	d_3	0.00 %	0.01 %
	ATT	37.52 min.	**36.66** min.
	ATT_{wop}	35.68 min.	**34.23** min.
	C_O	2378 min.	6173 min.
I-Operator	d_0	48.62 %	9.31 %
	d_1	32.45 %	24.77 %
	d_2	18.93 %	31.22 %
	d_3	0.00 %	24.39 %
	d_4	0.00 %	8.66 %
	d_5	0.00 %	1.58 %
	d_6	0.00 %	0.07 %
	ATT	40.88 min.	55.54 min.
	ATT_{wop}	37.36 min.	45.37 min.
	C_O	1077 min.	**319** min.
II-Operator	d_0	46.97 %	8.47 %
	d_1	31.84 %	24.03 %
	d_2	21.19 %	32.84 %
	d_3	0.00 %	20.66 %
	d_4	0.00 %	10.43 %
	d_5	0.00 %	3.03 %
	d_6	0.00 %	0.54 %
	ATT	41.26 min.	55.96 min.
	ATT_{wop}	37.655 min.	45.37 min.
	C_O	1265 min.	**319** min.

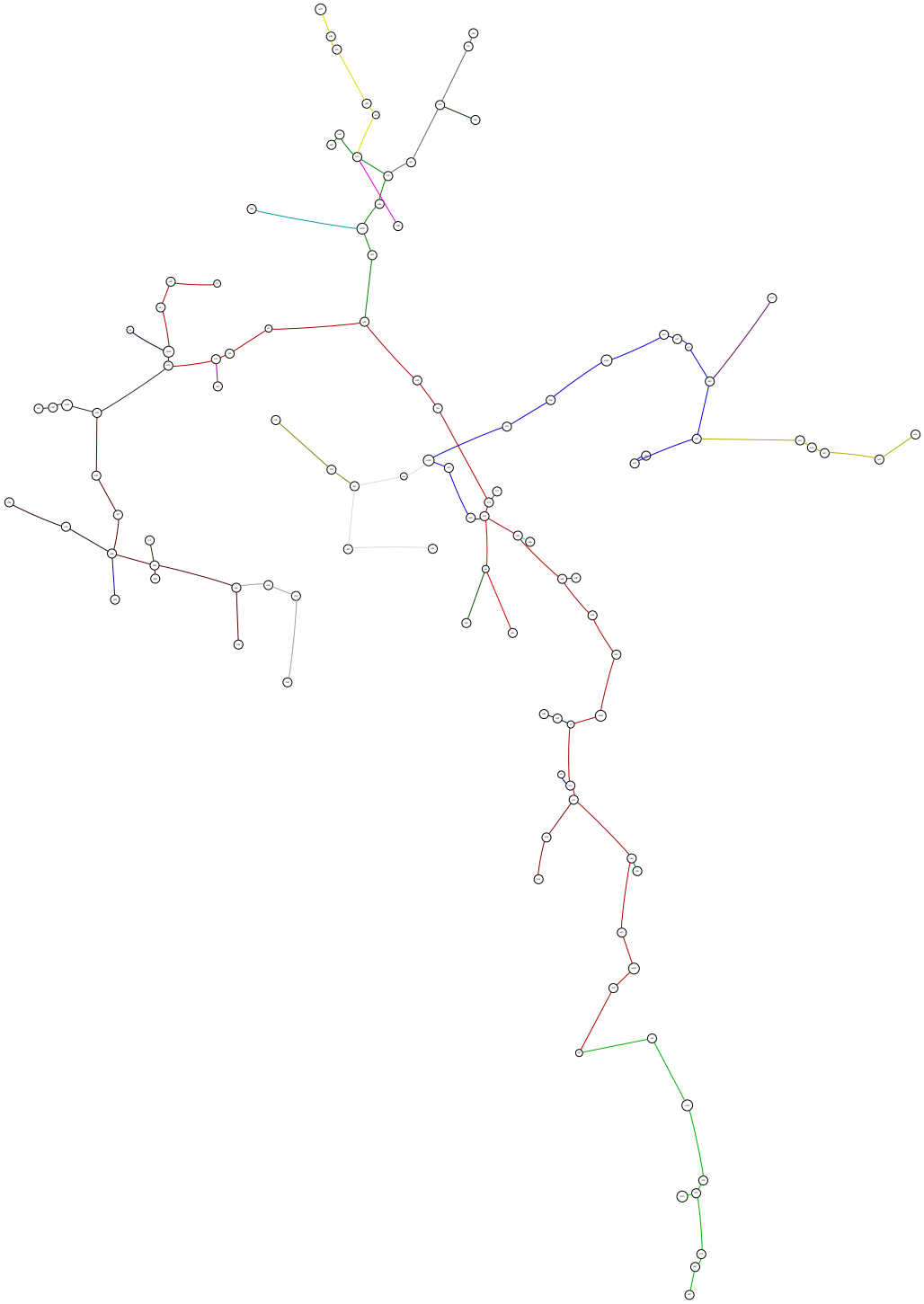

Figure 5.6: Operator-oriented route set for British city based network.

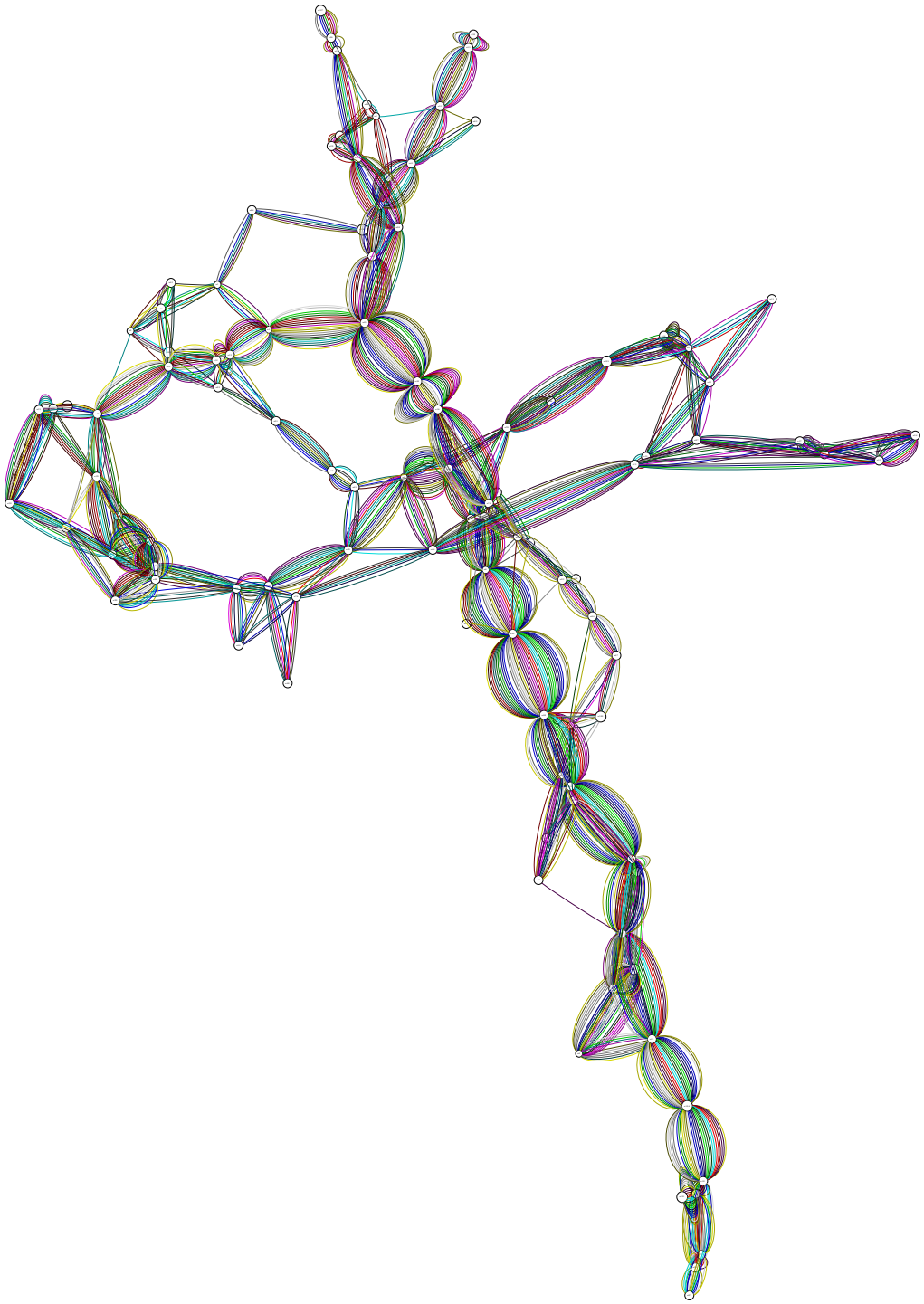

Figure 5.7: Passenger-oriented route set for British city based network.

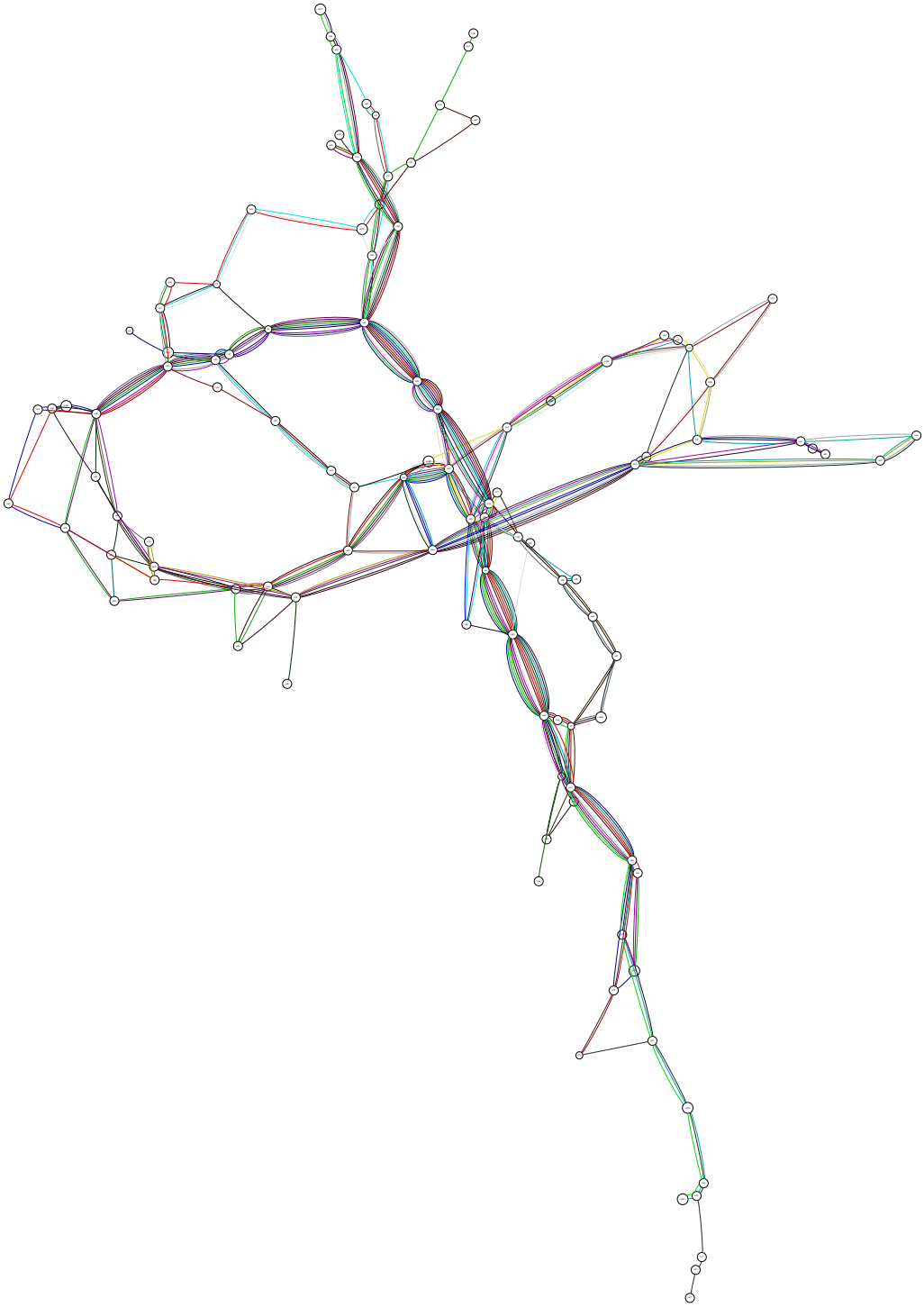

Figure 5.8: Balanced route set for British city based network.

et al. (2004). In their work, a list of previous results is published, together with new routes, that are evaluated as better than any other result at the time, and finally the authors provide quality factors for the given routes. But, when trying to assess the quality factors of their published routes with our implementations, both the MIP formulation and the genetic algorithm gave the same result, as expected, and it was different from what was published.

To try to explain this, differences in problem definitions were investigated. Nevertheless, we could not find any difference in the definition of how to evaluate a solution. We also experimented with making routes be unidirectional (instead of bidirectional), and this only generated results that were even farther from the published ones. Given that, we decided to utilize the quality factors given by both our MIP formulation and genetic algorithm in our comparisons. Since the implementations are completely independent, and they were able to reproduce every result from other works without problems, we consider them to be reliable.

We also perform comparisons with the work of Lang Fan *et al.* (2009) using two test cases, Mandl's network and a British city based network, as explained in this chapter. The results for Mandl's network were checked, since the route sets were published in detail, and every result was correctly reproduced. But, when analyzing the British city based network results, we perceived an inconsistency. Route sets were not published for this network, possibly because of space limitations, and this difficults validation.

To explain the inconsistency, we first recall the definition of the Average Travel Time without penalty (ATT_{wop}), which is how much time passengers would travel if the penalty was reduced from its current value, t_{pen}, to zero. To calculate this term, the following formula is used (where T_{MAX} is the maximum number of transfers): $ATT_{wop} = ATT - \sum_{i \leq T_{MAX}} t_{pen} d_i i$.

Recalling Section 2.2.3, the minimum travel time, if every route were available, is given by taking best paths in the *transport network* (where every edge is available). It is impossible to achieve a better result than the minimum travel time, since it already considers no penalties and uses the best paths, so no possible penalty configuration may result in smaller results than the minimum travel time.

The minimum travel time is known for the British city based network, and equals to 33.8395, as published by Lang Fan *et al.* and confirmed with our implementations. Nevertheless, the ATT_{wop} for the results of Lang Fan. *et al.* in passenger-oriented scenarios are 32.92, 33.36, 33.8365 and 33.85 minutes. Three out of four of the given $ATTs_{wop}$ are better than would be possible given the edge distances in the network, indicating that the published results do not correspond to legal results for the given instance and problem statement.

In order to be able to use their results for comparison, we assume that the authors were not considering penalties when calculating their $ATTs$, or in other words, they forgot to apply the penalty. Nevertheless, the other quality indicators also do not have great resemblance to normally found results with our algorithms, presenting combinations of very cheap route sets with very high covered demand. We present these results in detail in Section 5.2.2.

6 CONCLUDING REMARKS AND FUTURE WORK

The chosen problem definition was successful in allowing comparison with many previous works, as it proved generic and flexible enough to be compatible with them. We expect solutions to other, more specific problem statements, e.g. more realistic representations of the UTRP, to be able to take advantage of the techniques and algorithms developed in this work, with some adaptation.

We developed and implemented an exact MIP formulation for the UTRP. To the best of our knowledge, a full exact solution to this problem was never published before. With it, we obtained the best possible solutions for small sized scenarios, which were also never published before. A time estimate for exactly solving a certain instance of the problem was derived, and with it, limits on the problem size to which exact solutions are feasible can be derived. We also proposed a *divide-and-conquer* heuristic using the MIP formulation for each step. In addition, from the MIP formulation, we were able to obtain route suggestions for our metaheuristic approach. Finally, a formulation for redesigning an existing network was also given, which may be useful in practice to consider the cost and utility of changing existing routes.

We also proposed and implemented a genetic algorithm to solve the UTRP. With this approach, we achieved better results than every other known to us. Most of our solutions also *dominated* previous solutions, and thus did not only improve the main goal, but all goals.

This was done with a more flexible implementation in comparison to previous approaches, which is not bound to certain restrictions such as fixed number of routes, minimum and maximum number of nodes per route, allowance of cycles, directedness of routes and more. This characteristic is also due to the problem definition, which is more generic than previous ones.

The proposed approach uses the following ideas to reach better results: it carefully selects initial solutions from many different sources, including MIP relaxation, minimum spanning tree, shortest paths and from greedy algorithms; it applies *simplification* to prevent unnecessary routes; and it uses operators such as *exchange* and *crossover*, that exchange characteristics between routes and route sets.

Regarding reliability and correctness, results of previous works were reproduced and tested using both the MIP formulation and the genetic algorithm implementation, which have no relation between them (except of being written by the same author), and the results were coherent. We also extended our implementation to allow state saving at

some point of the computation, so as to enable division of the work in different time periods and machines. Another extension provides capability of transferring solutions from the genetic algorithm to the MIP formulation and vice-versa, what is useful for testing or optimizing parts of the problem. The programs can also export the networks in the Graphviz format (ELLSON et al., 2002) for easier visualization and debugging.

To test the proposed algorithm and implementation with further test cases, also considering real scenarios, we are currently using networks representing big capitals such as Porto Alegre, Brazil and Berlin, Germany in our tests. One difficulty in doing this is obtaining the demand data, i.e. the OD matrix. We can roughly estimate demands according to common sense, thus obtaining approximate results, but this does not allow direct comparison with the existing route set of the public transport system, since differences in quality factors may be due to differences in the OD matrix.

We are also integrating our implementation with simulation software in order to further validate and assess the used techniques and algorithms. The simulation package in use is MATSim (BALMER et al., 2009). An idea here is to substitute our quality factors by the evaluation function of the simulation package. In case of success, this would prove the generality of the methods in use here, and how they can adapt to different problem definitions and conditions.

Regarding performance, our genetic algorithm implementation has room for improvement. By simultaneously evolving different members of the population, the algorithm may be sped up by a factor equal to the size of the population, which can be set arbitrarily high, as long as respecting memory boundaries. This would be convenient in order to faster explore the search space, and would be essential when dealing with very big instances. Implementing this involves coordinating the use of many cores or machines, but does not require any significant change to the algorithm structure.

With the work developed here, we expect to improve the overall quality of methods and algorithms available to solve the UTRP, which becomes vital as public urban transit networks grow larger and more complicated.

REFERENCES

AGRAWAL, J.; MATHEW, T. V. Transit Route Network Design Using Parallel Genetic Algorithm. **Journal of Computing in Civil Engineering**, [S.l.], v.18, n.3, p.248–256, 2004.

ÁLVAREZ, A. et al. A computational tool for optimizing the urban public transport: a real application. **Journal of Computer and Systems Sciences International**, [S.l.], v.49, p.244–252, 2010.

BAAJ, M.; MAHMASSANI, H. An AI-based approach for transit route system planning and design. **Journal of Advanced Transportation**, [S.l.], v.25, n.2, p.187–209, 1991.

BALMER, M. et al. MATSim-T: architecture and simulation times. **Multi-agent systems for traffic and transportation engineering**, [S.l.], p.57–78, 2009.

BORNDöRFER, R.; GRöTSCHEL, M.; PFETSCH, M. E. A Column-Generation Approach to Line Planning in Public Transport. **Transportation Science**, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v.41, n.1, p.123–132, Feb. 2007.

CHAKROBORTY, P. Optimal Routing and Scheduling in Transportation: using genetic algorithm to solve difficult optimization problems. **Directions-Indian Institute of technology kanpur**, [S.l.], 2004.

CHAKROBORTY, P.; WIVEDI, T. Optimal route network design for transit systems using genetic algorithms. **Engineering Optimization**, [S.l.], v.34, n.1, p.83–100, 2002.

CPLEX, I. High-performance software for mathematical programming and optimization. **U RL http://www. ilog. com/products/cplex**, [S.l.], 2009.

CURTIN, K. M.; BIBA, S. The Transit Route Arc-Node Service Maximization problem. **European Journal of Operational Research**, [S.l.], v.208, n.1, p.46 – 56, 2011.

DIJKSTRA, E. A note on two problems in connexion with graphs. **Numerische mathematik**, [S.l.], v.1, n.1, p.269–271, 1959.

ELLSON, J. et al. Graphviz—open source graph drawing tools. In: GRAPH DRAWING. **Anais...** [S.l.: s.n.], 2002. p.594–597.

FAN, L. **Metaheuristic Methods for the Urban Transit Routing Problem**. 2009. Tese (Doutorado em Ciência da Computação) — PhD. dissertation, Cardiff University, Cardiff, United Kingdom.

FAN, L.; MUMFORD, C. L. A metaheuristic approach to the urban transit routing problem. **Journal of Heuristics**, Hingham, MA, USA, v.16, n.3, p.353–372, June 2010.

FAN, L.; MUMFORD, C. L.; EVANS, D. A simple multi-objective optimization algorithm for the urban transit routing problem. In: ELEVENTH CONFERENCE ON CONGRESS ON EVOLUTIONARY COMPUTATION, Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2009. p.1–7. (CEC'09).

FAN, W.; MACHEMEHL, R. B. Optimal Transit Route Network Design Problem with Variable Transit Demand: genetic algorithm approach. **Journal of Transportation Engineering**, [S.l.], v.132, n.1, p.40–51, 2006.

FAN, W.; MACHEMEHL, R. B. A Tabu Search Based Heuristic Method for the Transit Route Network Design Problem. In: HICKMAN, M.; MIRCHANDANI, P.; VOSS, S. (Ed.). **Computer-aided Systems in Public Transport**. [S.l.]: Springer Berlin Heidelberg, 2008. p.387–408. (Lecture Notes in Economics and Mathematical Systems, v.600).

FLOYD, R. W. Algorithm 97: shortest path. **Commun. ACM**, New York, NY, USA, v.5, n.6, p.345–, June 1962.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. [S.l.]: University of Michigan Press, 1975. 1–200?p. v.Ann Arbor.

HWANG, F.; RICHARDS, D. Steiner tree problems. **Networks**, [S.l.], v.22, n.1, p.55–89, 1992.

ISRAELI, Y.; CEDER, A. Designing transit routes at the network level. In: VEHICLE NAVIGATION AND INFORMATION SYSTEMS CONFERENCE, 1989. CONFERENCE RECORD. **Anais...** [S.l.: s.n.], 1989. p.310 –316.

KATZ, G.; KIDER JR, J. All-pairs shortest-paths for large graphs on the GPU. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON GRAPHICS HARDWARE, 23. **Proceedings...** [S.l.: s.n.], 2008. p.47–55.

KIDWAI, F. **Optimal design of bus transit network**: a genetic algorithm based approach. 1998. Tese (Doutorado em Ciência da Computação) — PhD. dissertation, Indian Institute of Technology, Kanpur, India.

KRUSKAL, J. On the shortest spanning subtree of a graph and the traveling salesman problem. **Proceedings of the American Mathematical society**, [S.l.], v.7, n.1, p.48–50, 1956.

MANDL, C. Evaluation and optimization of urban public transportation networks. **European Journal of Operational Research**, [S.l.], v.5, n.6, p.396–404, 1980.

MAZLOUMI, E. et al. Efficient Transit Schedule Design of timing points: a comparison of ant colony and genetic algorithms. **Transportation Research Part B: Methodological**, [S.l.], v.46, n.1, p.217 – 234, 2012.

MILLER, C.; TUCKER, A.; ZEMLIN, R. Integer programming formulation of traveling salesman problems. **Journal of the ACM (JACM)**, [S.l.], v.7, n.4, p.326–329, 1960.

SHIH, M.; MAHMASSANI, H. Vehicle sizing model for bus transit networks. **Transportation Research Record**, [S.l.], n.1452, 1994.

TOM, V. M.; MOHAN, S. Transit Route Network Design Using Frequency Coded Genetic Algorithm. **Journal of Transportation Engineering**, [S.l.], v.129, n.2, p.186–195, 2003.

VAZIRANI, V. **Approximation algorithms**. [S.l.]: Springer Verlag, 2001.

WAN, Q.; LO, H. A Mixed Integer Formulation for Multiple-Route Transit Network Design. **Journal of Mathematical Modelling and Algorithms**, [S.l.], v.2, p.299–308, 2003. 10.1023/B:JMMA.0000020425.99217.cd.

ZHAO, F. Large-scale transit network optimization by minimizing user cost and transfers. **Journal of Public Transportation**, [S.l.], v.9, n.2, p.107, 2006.