School of Economics and Management

Tilburg University

Heuristic methods for the design of edge disjoint circuits.

An application to fiber rings in telecom networks.

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Operations Research and Management Science

Author:
R.C.J. Daamen
(ANR: 938789)

Supervisor Tilburg University:
Prof. Dr. Ir. E.R. van Dam
Supervisors TNO:
Drs. F. Phillipson RTD
Dr. D.T.H. Worm

August 20, 2013

Abstract

In this thesis heuristics are studied for the design of edge disjoint circuits. These are applied to a telecommunication network design problem: the roll-out of Fiber to the Cabinet. This problem setting can be represented by a graph. A small subset of the vertices are street cabinets and one vertex is the central office. The objective is to minimize the sum of the circuit costs to connect all street cabinets to the central office, while satisfying capacity restrictions and edge disjointness within each circuit. In the studied graphs, inexpensive paths are present with available ducts, which complicate the analysis considerably. The problem is NP-hard, since it is a generalization of the NP-hard Capacitated Vehicle Routing Problem. Therefore, an insertion heuristic is developed that uses an extension of the work of Suurballe and Tarjan (1984) to construct edge disjoint shortest paths. After the construction of an initial solution, local search is used to improve the solution. In previous work on network design problems and on vehicle routing problems, this algorithm of Suurballe and Tarjan (1984) has never been used. There the focus was on exact approaches that take very long to solve for large instances or on heuristics that use routing techniques that quite often lead to high routing costs. Finally, the performance of the insertion heuristic is compared to the cluster first - route second heuristic developed by TNO. The insertion heuristic clearly shows better performance for almost all types of instances that were studied. Especially for large instances, the cluster first - route second heuristic is completely outperformed.

Contents

1 Introduction

1.1 Problem description

In the last decennium internet broadband usage has increased a lot. More and more services use internet nowadays and especially digital services that include video (which requires much bandwidth) are emerging fast. This trend will continue in the coming years; according to a conservative estimate by TNO (2010) bandwidth use in the Netherlands will increase by approximately $30 - 40\%$ per year until 2020 (see illustration in Figure 1). This leads to an estimated average download speed in the range of 75Mbit/s to 400Mbit/s in 2020. The growth can mainly be contributed to users that are going to use the internet more often and more intensively. In other words, the growth is mostly endogenous and not exogenous, because the penetration rate of internet in the Netherlands is already very high and not many new users are expected in the near future. The increase in HD video services and the popularity of cloud services are the most important reasons for the increase in bandwidth usage (TNO, 2013). At the same time consumers have more and more devices that are connected to the internet (smartphones, tablets, gaming consoles, televisions, etc.).

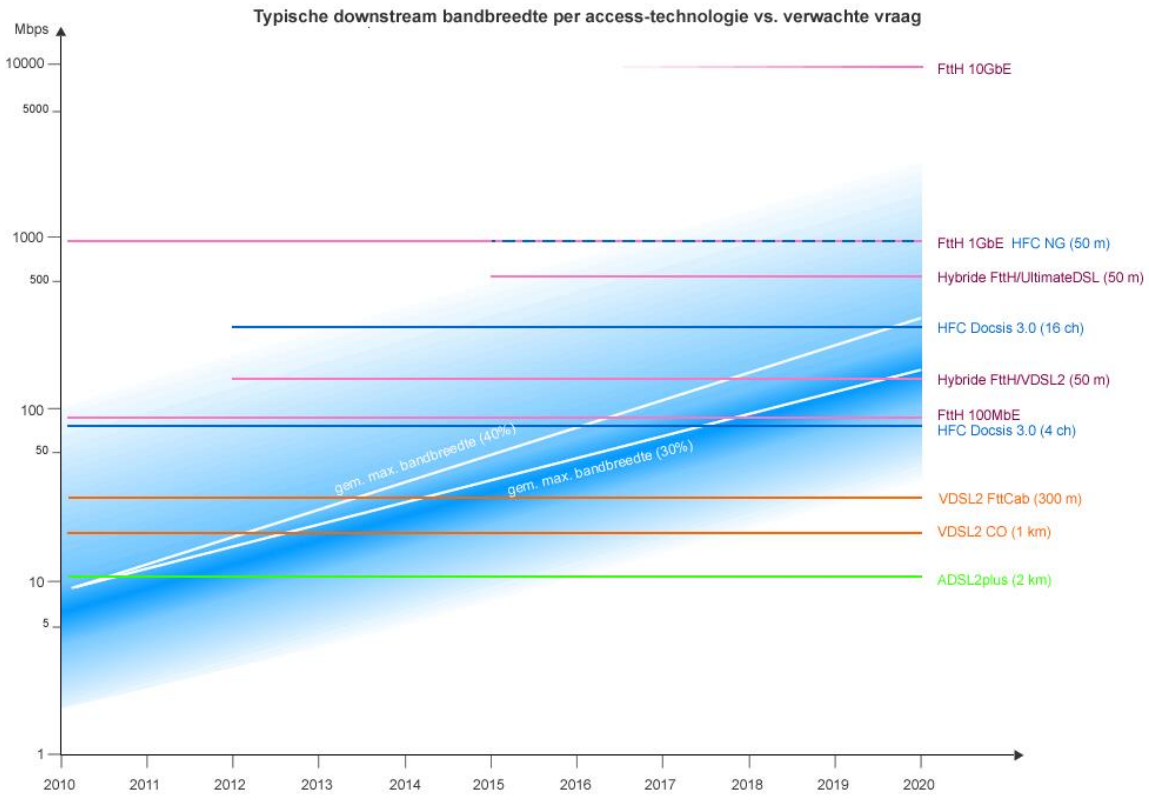


Figure 1: Expected increase in bandwidth demand over 2010-2020 and the limitations of the various techniques (TNO, 2010).

Broadband internet is offered in different ways in the Netherlands. Firstly, the fiber-copper

network (DSL) is used to connect customers to the internet. This network originates from the former telephone network and has almost 100% of the households in the Netherlands connected to it (TNO, 2013). This network is operated by KPN and is additionally used by some other internet service providers (ISPs). This network will be studied in this thesis. Secondly, internet is offered over the cable network that has a 99% penetration rate into the homes in the Netherlands (TNO, 2013). The main ISPs owning this network are Ziggo and UPC. Each owns and operates another part of the total network and the network is not open to other ISPs, unlike the DSL which is. Thirdly, 18.4% of the households had access to a fiber network in 2012 (TNO, 2013). A major fiber network owner is Reggefiber, which gives various ISPs access to its network. Figure 2 shows the availability of Fiber to the Home (FttH) in the Netherlands in April 2013. Lastly, also mobile internet is offered via e.g. 3G and in the future 4G. The latter can in the future be used for sparsely populated rural areas for which it is too expensive to roll-out fiber (close) to the homes. The telecom network is currently not able to meet

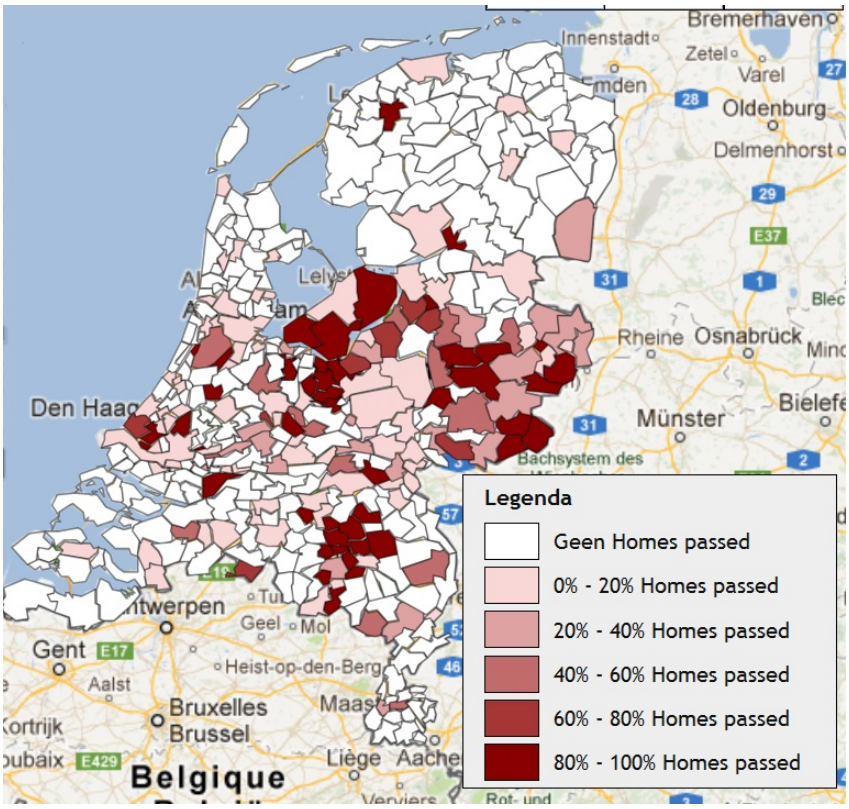


Figure 2: FttH availability in the Netherlands in April 2013 (Stratix, 2013).

the strongly increasing demands and has a hard time competing with the cable operators who are still able to offer the required bandwidths at the moment (Phillipson, 2013a). Therefore, a migration towards a topology is necessary where a smaller distance has to be bridged over copper; in other words fiber should be brought closer to the homes. This migration can take place in different forms: it can either take place at once or in steps. In Figure 3 four different topologies are illustrated. The first topology illustrates the current situation that is still present in the largest part of the Netherlands. The migration to the fourth topology full FttH (Fiber to the Home) can either be done at once or in intermediate steps: FttCab (Fiber to the Cabinet) and/or Hybrid FttH (Hybrid Fiber to the Home). The first intermediate step, FttCab, uses either the VDSL technique or the G.Fast technique. The latter is not yet available to the market and the 4GBB project in which many parties are involved (including TNO) investigates this

technique Brink (2011). The Hybrid FttH brings the fiber closer to the customer's premises, e.g. to the street or the basement of a flat. Also here the G.Fast technique is used.
Phillipson (2013a) presented a case study for which it is shown that the migration with two intermediate steps is favorable to the direct migration in terms of costs. The migration from the Full Copper (FC) network to the FttCab topology is the subject of this thesis. The methods developed in this thesis are, however, also useful for the migration from FttCab to Hybrid FttH and subsequently to Full FttH.

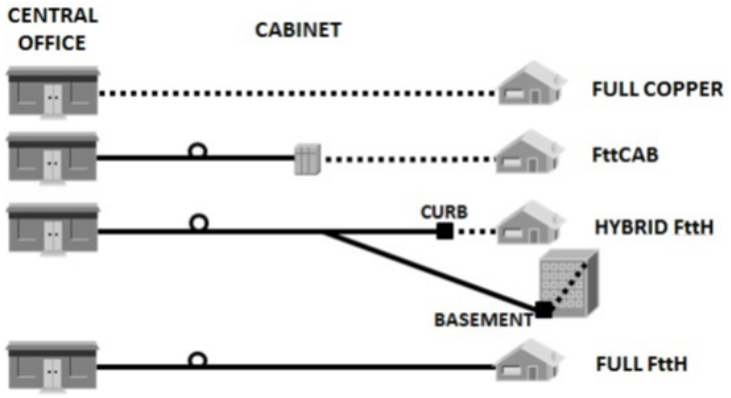


Figure 3: Four different topologies for the telecom network.

A phased roll-out has two main advantages (Phillipson, 2013a):

1. FttCab can be realized much quicker, because less digging is necessary compared to connecting every single home with fiber. In this way the growing bandwidth demand can be accommodated (see Figure 1) and the telecom operator can compete with the cable operators. Subsequently, a couple of years later the step can be made to Hybrid FttH when bandwidth demand has grown to a level where the FttCab is no longer sufficient. If the telecom operator would not choose for a phased migration, it is likely to lose many customers in large parts of the country where the customer's bandwidth demand exceeds what it can deliver. In fact, the number of households that can migrate to FttH yearly is restricted (e.g. due to workforce restrictions and municipalities not allowing too much inconvenience).

2. When in the future this higher supplied bandwidth turns out to be insufficient, the remaining part of the connection to the homes can be installed using (most) of the earlier installed cables and equipment. Some extra investment costs are incurred in the phased migration, as some investments in technology needed in the FttCab and Hybrid FttH are lost. Phillipson (2013a) estimated the investment costs for a direct and a phased migration (with two intermediate steps) to FttH for a case study. As expected, the costs are higher in the phased migration, but since in the phased migration some investments take place later these need to be discounted. Using the weighted cost of capital (WACC) for fixed telecom operators from Mason (2010) as a discount rate, Phillipson (2013a) calculated for a Dutch case an approximately 15% lower total discounted cost for the phased migration than for a direct migration to full FttH.

To give the reader an insight in the order of magnitude of the required investments, note that the one-step transition to full FttH has an estimated total cost of 1025-1375 euro per connection (FttH Platform Nederland, 2012). Considering that the Netherlands has in total around 7.5

million households (CBS, 2012), for a complete roll-out FttH a total investment between nearly 8 to more than 10 billion euros is necessary. In that light it might be very beneficial to consider a phased migration, since the study by Phillipson (2013a) shows that this has great potential. However, some further investigation is necessary to draw a conclusion for the Netherlands as a whole, since there is likely to be a difference in costs between more and less densely populated areas.

Within this migration trajectory, several factors influence the total costs. Among them the distance over which the fiber needs to be installed is a very important one. In that light, good algorithms are necessary that determine a very cost-efficient clustering of the street cabinets and optimize the routes of fiber, i.e. under which streets should the fiber cables be placed to connect the street cabinets. The combination of these two problems is the research topic of this thesis. The exact description of the problem is discussed next.

1.2 Problem definition

The problem studied in this thesis deals with the design of a telecommunication network. The topic of interest is the roll-out of Fiber to the Cabinet (FttCab) for the network of a telecom operator. The design of this network has to satisfy certain properties and restrictions:

1. A ring-shaped network structure is used to guarantee a high reliability. Ring-shaped networks have the advantage that in case a disruption occurs somewhere in the network, the network does not go down completely. When a disruption occurs in a non-ring-shaped network all users downstream from the disruption are not connected anymore.

2. At least a certain percentage of the households, e.g. 85%, should be connected to the fiber network within a specified distance, e.g. 1 km. To ensure that this restriction is satisfied, enough street cabinets have to be 'activated', i.e. they have to be connected to the central office (CO) via fiber. The problem which street cabinets to activate was solved by Phillipson (2013c) and is not part of the research in this thesis. The street cabinets that need to be activated are considered given in this thesis.

3. There is a maximum number of customers that can be included in one circuit. Every activated street cabinet has a given number of customers that are connected to it. Several clusters of street cabinets have to be formed. For each cluster the restriction has to be satisfied that it has not more than the maximum number of customers included in it.

4. Each street cabinet will be included in only one cluster. This is a logical property of the network design, since it would just be too expensive to include every street cabinet in multiple circuits.

5. To connect the street cabinets and the central office in a cluster by a ring there is given a network of edges and nodes that can be used. The edges can be divided into two main types: edges with already available pipes that are not used to full capacity yet and edges without preexisting pipes or where the pipes are completely utilized, but where (extra) excavations are permitted. No major excavations are necessary in the former case, whereas in the latter a significant part of the costs are due to excavations.

6. Each ring that is constructed should be edge disjoint. In other words, it is not allowed that an edge is used twice within one ring. The edge disjointness concerns the most detailed level, i.e. street level. In other words, a pipe in a certain street cannot be used twice in the same ring. Note further that between rings, no edge disjointness is required. Multiple rings can, therefore, make use of the same pipe. Another remark to be made is that a ring is not required to be node disjoint.

This problem has been investigated before by TNO (i.a. described in Phillipson (2013a), Phillipson (2013b) and Phillipson (2013c)). They use a cluster first - route second approach, i.e. first the street cabinets are clustered and only afterwards is determined where the fiber cables are to be put (below which street). The clustering is based solely on minimizing the distance between the street cabinets. These distances are not based on the street pattern or the ditches that can be used to put the cables in, but are as the crow flies (a straight line). This has the advantage that it is easy, but at the same time it has an important disadvantage. The chosen clusters may be cost-inefficient when taking into account the real street patterns and ditching restrictions. The disjoint routing is done in a very simple way, which can lead to solutions of very bad quality or in the worst case even to no solution at all (while there does exist one).

The objective of the research of this Master's thesis is to develop a method that integrates the cluster and routing decision in one to avoid the above mentioned disadvantage. Moreover, the routing problem is studied in detail to see if improvements are possible there. The computation time of the method is targeted to be in the order of magnitude of minutes. This is the reason that the focus is on the development of heuristic methods, since exact methods will exceed the preferred computation time very soon as instances grow in size. It may seem counterintuitive that short computation times are preferred over very high solution quality, since the planning takes place only once and not an continuous basis. Often in such situations computation time is of less importance. However, in this case the situation is a bit different. Firstly, network planners prefer a tool that is able to find a good solution quickly. Not all practical constraints and decisions can be included in the model and the data will not be 100% reliable, so in that sense it is much more important to get a good solution quickly than an optimal solution that took days or even weeks to calculate. Some adaptations will need to be made to the solution anyway. Ideally, planners could use the tool to play around a bit with the solution to come to a really practically implementable network design plan using their knowledge and experience. Furthermore, no expensive solvers such as CPLEX can be used due to strict software regulations at network operators. This also prevents solving the problem in an exact fashion.

The problem studied in this thesis is NP-hard, since it is a generalization of the Capacitated Vehicle Routing Problem (CVRP), which is shown to be NP-hard (Lenstra and Rinnooy Kan, 1981). It is capacitated, since only a limited number of street cabinets can be combined in a circuit. Additionally the underlying street pattern and the restriction that a ditch cannot be used twice has to be taken into account (restriction 6 above). This prevents applying the solution methods of the CVRP directly. Most other main known extensions of the VRP (e.g. time-windows, periodic, split delivery, pick-up and delivery, backhauls, etc.) are not relevant to this problem. Only the multi depot VRP could be relevant in the sense that it might be possible that street cabinets can be connected to different central offices. The choice to which central office a street cabinet should be connected will, however, not be part of this research.

In the next section the related literature is reviewed. In Section 3 the problem outlined in this section is described more formally and some prerequisites methods are discussed. Next, in

Section 4 two approaches are discussed to construct an initial feasible solution to the problem: the original approach by TNO and the new integrated approach. In Section 5 procedures are discussed that can try to improve the initial solution. Then in Section 6 the instances are described that are used to test the developed methods. In this section also an example is given to illustrate the methods. Subsequently, in Section 7 the test results are outlined. Finally, in Section 8 the conclusions of this thesis are given and suggestions are made for further research.

2 Literature overview

2.1 Network design problems

In this section network design problems that are related to the one studied in this thesis are outlined. This section is meant to illustrate some known problems that relate closely to the problem studied in this thesis, while especially indicating the differences with this thesis' research. For a completer overview, the interested reader can consult e.g. Laporte and Martín (2007) or Beasley and Nascimento (1996). This section is solely devoted to ring networks for several reasons: this type of network is also used in this thesis and more importantly, it is this type of network that is often employed in telecommunication networks to be able to guarantee a high reliability. When a disruption occurs somewhere in the network, the network does not go down, since each node is two-connected to the depot. The disadvantage of a tree network is namely that in case of a failure on a node or an edge the entire downstream network goes down. Most attention in this section is paid to the Ring Network Design Problem and the Capacitated m-Ring-Star problem, since these are closest to the problem studied in this thesis. Kalsch et al. (2012) study the same problem. Their approach has, however, some disadvantages. These are described in the next section. For all other papers it holds that some clear differences prevent direct application of the methodology from that research area to the problem in this thesis.

2.1.1 Ring Network Design Problem

Gendreau et al. (1995) and Fink et al. (2000) formulate heuristics for the (General) Ring Network Design Problem. This problem deals with the construction of one single ring. Revenues are defined for each pair of nodes included in the ring and construction costs for each direct link in the ring are used. The objective is to maximize the sum of all revenues minus the construction cost when building the ring. The problem deals with selecting the best nodes to include in the ring while possibly satisfying some constraints, e.g. a budget constraint or a maximum number of nodes that can be included in the ring. Although this problem deals with the construction of a ring through nodes, is quite different from the problem studied in this thesis. Firstly, in this thesis' problem several rings can be constructed instead of just one. Secondly, no revenues for including a street cabinet are assumed here and no budget constraints are taken into account. Moreover, which nodes need to be included in the rings is assumed to be given in this thesis. Furthermore, the complicating issues around preexisting pipes and edge disjointness are not considered in the Ring Network Design Problem.

The problem studied by Henningsson et al. (2006) is similar to the problem studied in this thesis, but some subtle, nevertheless important, differences are present. They deal with the construction of edge disjoint rings covering all nodes at least once. Also included is the constraint on the maximum number of connected nodes in a ring. Although they require edges of the ring to be disjoint, this does not at all prohibit using a duct multiple times as is desirable from reliability perspective. They do not consider a street or duct pattern, so edge disjointness is not necessarily satisfied on the detailed level that is studied in this thesis. Henningsson et al. (2006) present a mathematical programming formulation of the problem and a method based on column generation. The computation times for instances with up to 100 nodes are up to half an hour. The intended computation times for the methodology in this thesis are much smaller,

since instances of up to 600 nodes need to be solved in a short time. The problem studied is also much more difficult, because a graph of street/duct detail needs to be used to guarantee disjointness of the edges in a ring.

Kalsch et al. (2012) develop a mathematical model and a heuristic that looks how to embed a ring structure in a fiber network. This setting is similar to the one studied here in that in both cases some existing network is considered. The interpretation is only a little bit different: fiber cables versus ducts (yet without fiber cables). First they develop a mathematical model that takes into account the following restrictions: ensuring a ring structure, a maximum number of nodes in a ring, each node in exactly one ring, and that the ring uses each edge only once. Apart from some small differences, the mathematical model developed in Kalsch et al. (2012) is almost directly applicable to the problem studied in this thesis. However, no computation times are mentioned in the paper for this model, which might indicate they they are too large. They quickly continue with the development of a heuristic based on a decomposition and solving multiple integer programming models. The description of this heuristic is quite brief and some details are not given. For example, the heuristic is said to first determine the assignment of nodes to rings, but it is only remarked that this could be done by looking at neighboring nodes. After that it is immediately assumed that a set of nodes is given which has to be included in each ring. Then an integer linear program is presented that can be used for each subproblem (each ring) to find out how the ring should be constructed optimally. There is no guarantee that a ring has no edges that partly overlap when considering the real ducts. They do consider existing (cheap) ducts; they mention that this can taken into account by weighting the cost parameter in the objective of the model.

Next, they build a heuristic based on iteratively applying this submodel that does prevent any overlap in the used edges in a ring. Two different type of edges are distinguished: the 'ring edges' and the 'cable edges'. A ring comprises of ring edges, which in turn consist of (multiple) cable edges. Therefore, it can happen that a cable edge is used multiple times in a ring in different ring edges. This is undesirable, since when it fails part of the network is down. When it is ensured that a cable edge is in only one ring edge this does not happen in case of a failure, since all nodes are still connected to the depot because of the ring structure. When this is the case, the problem is said to satisfy the shared risk link condition. The iterative heuristic is then defined as follows: first run the previously defined model that does not take into account the cable edges. Then check if the shared risk link condition is satisfied, i.e. check if there are indeed no cable edges used twice in the ring. If the condition is not satisfied start an iterative procedure where in the kth iteration the first k ring edges are fixed. Then the submodel is ran, extended with a penalty in the objective on each shared risk cable. The procedure stops if the shared risk condition is satisfied or in the worst case after considering all ring edges. Finally, also they report that the solution can be further improved by using solution polishing, a tool from CPLEX. This is not further explained in their paper. Kalsch et al. (2012) report only one test result: in a graph consisting of $13,246$ cable nodes and $22,116$ cable edges, 135 rings were constructed using the heuristic in 4.8 hours using a computer with decent specifications. It is, however, hard to draw conclusions on the performance of their approach, since no further information is given on the data than is mentioned over here. The goal of this thesis is, however, to come up with a method that is much faster: being able to solve real-life cases in the order of magnitude of minutes instead of hours. From practical perspective, it is also not desirable to use expensive commercial software, such as the CPLEX used by Kalsch et al. (2012), to perform the analysis. Another important disadvantage of their method is that no real attention is paid to the clustering of the nodes to the rings. This has namely a non-negligible effect on

the overall solution. In this thesis this decision is explicitly considered simultaneously with the routing decision.

2.1.2 Capacitated m-Ring-Star Problem

The Capacitated m-Ring-Star Problem was introduced by Baldacci et al. (2007). The Ring Star Problem (RSP) without the capacity constraint was already introduced earlier by Labbé et al. (2004). The CmRSP deals with the design of rings (circuits) that connect customers to a central depot. Customers can either be situated on the ring itself or they can be directly connected to a visited node on the ring. In this way, a ring-star network is formed. Restrictions taken into account are that the rings need to be node-disjoint (for reliability) and that each ring has a maximum number of customers that it can include. Furthermore, next to the customers, transit points (Steiner nodes) are considered, which are intermediate points that can be included in a ring, but do not necessarily have to. Because of the requirement that the rings need to be node-disjoint, each transit point can at most be included in one ring. The objective in the CmRSP is to minimize the sum of two cost types: routing costs (costs of the ring) and allocation cost (cost of connecting customers to the ring).

The CmRSP differs from the problem studied in this thesis in several ways. Firstly, in the CmRSP it is implicitly assumed that no preexistent pipes are available which can be used against significantly lower cost compared to new pipes (Baldacci et al., 2007). In this thesis these preexistent pipes are explicitly taken into account, since in quite a lot of situations they are in fact available. Naji-Azimi et al. (2010) recognize the importance by stating that the cost of excavations to lay down the pipes is the most important cost, but at the same time they do not take these existing pipes explicitly into account, because in their case study there were no. Including discounts based on availability of pipes into their data might be possible. However, this is not so easily done since it might lead to ending up with a star-shaped network instead of the from a reliability perspective required ring network. Situations in which no or very few pipes are available (e.g. when developing new districts, in fast growing cities or in areas where a telecommunication network is barely present) are not explicitly studied in this thesis, but the methodology of this thesis can be applied to these situations, since it is a special case of the more general problem studied here.

Secondly, in the problem in this thesis no explicit end-customers are considered, but the 'customers' are in fact street cabinets. These street cabinets are in turn connected to the households. In that light the problem becomes harder, since the decision which 'customers' to include in the ring and which to connect to the ring is no longer solely based on cost considerations. Restriction 2 from the problem definition in Section 1.2 explicitly states that a certain percentage of all households needs to be within a specified distance from fiber. As explained there, the problem which street cabinets to activate, is for this reason not part of this thesis' research. The part of the network that is studied, is therefore ring-shaped instead of ring-star-shaped. This simplifies the CmRSP to some degree, but instead the complicating factor of the preexistent pipe network is included together with the restriction that a pipe can only be used once.

After the introduction of the CmRSP by Baldacci et al. (2007) several papers have investigated the problem. Some of them focus on exact solution methods, whereas other choose a heuristic approach. Exact formulations (IP formulations and branch and cut algorithms) are e.g. given

by Baldacci et al. (2007) and Hoshino and de Souza (2012). For medium-sized instances the computation time, however, already explodes. Large real-world instances are, therefore, not really solvable in a reasonable computation time by exact methods. Mauttone et al. (2008) developed a heuristic based on constructing an initial solution and then using local search to improve the solution. In particular, they use Tabu Search to avoid repeatedly swapping the same nodes and shaking to add extra diversification. Naji-Azimi et al. (2010) also developed a heuristic for the CmRSP. Their heuristic has a similar structure as Mauttone et al. (2008): an initialization procedure, improvement procedure and shaking. They show, however, that their heuristic outperforms that of Mauttone et al. (2008) both in computation time as in solution quality. Their construction of the initial solution uses the following idea: first define m clusters by adding only the depot and one node in each cluster. Make the first cluster by considering the depot and add the node furthest away from it. The second cluster is formed by the depot and the node furthest away from all nodes in the first cluster. The first node in the third ring is the node furthest away from the nodes in the first and the second cluster, etc. This idea is also investigated for the problem in this thesis. After defining the first nodes in all the m clusters, the rest of the nodes is assigned to the best position in the rings of the different clusters. How this is done exactly is not given in the paper. The improvement procedure of Naji-Azimi et al. (2010) swaps nodes within a ring, removes Steiner-nodes, and extracts and reassigns nodes to new rings. Finally, the shaking is implemented by extracting and reassigning a fixed number of nodes when the current solution could not be improved anymore by the improvement procedures.

Naji-Azimi et al. (2012) again propose a heuristic for the CmRSP. This time it is a matheuristic, i.e. it integrates mathematical programming in a heuristic. It builds for a large part on their previous heuristic from 2010 and the most important difference is the repeated use of a integer linear programming model within the heuristic. The results in Naji-Azimi et al. (2012) are superior to Naji-Azimi et al. (2010), but this comes at the cost of longer computation times, especially for the larger instances.

2.1.3 Multi-Depot Ring Star Problem

The just discussed CmRSP also has a variant with multiple depots: the Multi-Depot Ring Star Problem. It was introduced by Baldacci and Dell'Amico (2010). Unlike in the CmRSP, a choice still has to be made which customers to connect to which depot. The rest of the problem is the same. Baldacci and Dell'Amico (2010) develop heuristics to solve this problem. They give two options to construct a solution and then use tabu search to improve the solution. The first construction algorithm is based on the idea of route first - cluster second?, since it first assumes that there is only one depot and then makes rings. Afterwards, this artificial depot is removed from the rings and each ring is connected to the cheapest depot. The second construction algorithm on the other hand is based on cluster first - route second?. It first assigns the customers to depots and then for each depot a heuristic for the CmRSP is used. After construction of an initial solution tabu search is used to improve the solution. Baldacci and Dell'Amico (2010) define three possible neighborhoods that can be used. Their testing showed that a neighborhood that allows infeasible solutions, but does put a Lagrangian penalty term on it, performed best for their instances. The Multi-Depot Ring Star Problem has the same differences in problem description as this thesis has compared to the CmRSP, since the CmRSP is only extended by the decision which depot to assign each node to. This is also a

possible extension of the problem in this thesis.

2.1.4 Non-Disjoint m-Ring-Star Problem

Fouilhoux and Questel (2012) study a problem called the Non-Disjoint m-Ring-Star Problem (NDRSP). They propose an mixed-integer linear program and a branch and cut algorithm for the problem that differs from the CmRSP in that each edge is allowed to be used a maximum number of times. This is motivated by the authors from the assumption that each edge contains multiple fibers that can be used by different rings. They do explicitly model that between each node and the depot there are two edge-disjoint paths. In this way it the ring-structure is ensured. By making some small changes, part of the mathematical model formulation from Fouilhoux and Questel (2012) could be used for an exact approach to the problem in this thesis. As also seen from their results, such exact approaches are not able to solve the large instances that are aimed for in this thesis in a reasonable time. This is the reason that in this thesis solely heuristics are investigated.

2.1.5 Ring Spur Assignment Problem

Another possible network design is the Ring Spur Assignment Problem (RSAP) investigated by Carroll et al. (2013) and Carroll and McGarraghy (2013). Both use a branch and cut approach for this problem that has local rings that are interconnected by a so-called tertiary ring. The tertiary ring visits at least one node from each ring, but is allowed to visit multiple. Additionally, a node can be connected to a local ring by a spur (a single-edge connection), e.g. when it is geographically not possible to include the node in a ring. The problem studied in this thesis is different from the RSAP in that in this thesis the decision which nodes to include in the rings is much more difficult and therefore solved separately in advance. Moreover, in this thesis preexistent pipes are available that can be used against low cost. This complicates the analysis significantly compared to Carroll et al. (2013) and Carroll and McGarraghy (2013), since ensuring that the solution is indeed an edge disjoint ring is a major point of attention in this thesis. Finally, in the RSAP another hierarchy level is present compared to the problem studied in this thesis: every node is connected to the central office via one ring, instead of via a higher-level tertiary ring.

2.2 Edge disjoint shortest paths

One important restriction mentioned in the problem definition in Section 1.2 is the edge disjointness of a ring. A ring consists of shortest paths between street cabinets that each consist of possibly several edges. Therefore, in this section relevant literature on the disjoint shortest paths literature is discussed.

Many papers have been published from the 1950s onwards that deal with all kinds of versions of disjoint path problems (see e.g. the surveys in Korte et al. (1990)). For some versions an algorithm is found that can solve the problem in polynomial time, whereas some other versions are proven to be NP complete. The versions differ e.g. in which paths should be disjoint,

in whether the number of disjoint paths is fixed, whether the paths need to be edge or node disjoint and the type of graph that is considered (e.g. undirected/directed/directed acyclic, and other properties like planarity, nonnegativity of the weights, etc.). Relatively few papers focus, however, on the optimization version of the problem (disjoint *shortest* paths) that is of interest for the research in this thesis. The literature on this disjoint *shortest* path problem is discussed next.

Early research efforts on the disjoint shortest paths problem include the work of Suurballe and Tarjan (1984) who develop a polynomial algorithm for the problem to determine two disjoint shortest paths between a pair of vertices (see Figure 4). The idea behind the algorithm is very similar to Dijkstra's algorithm and it therefore computes two disjoint shortest paths from an origin vertex to all possible destination vertices (asymptotically) as fast as for only one destination vertex. Another problem is finding two disjoint shortest paths from a single source vertex to two destination vertices t_1 and t_2 (see Figure 5). This problem can be solved by introducing an artificial sink vertex t' that is connected to the two destination vertices t_1 and t_2. Then the problem is reduced to finding two disjoint paths from source s to target t'. This can be solved using the Suurballe and Tarjan (1984) algorithm or by the perhaps more intuitive algorithm given by Bhandari (1999). These two algorithms are discussed in more detail in Section 3.3. Yang and Zheng (2006) formulated two algorithms for this problem that have a slightly better asymptotic complexity. They are inspired by the work of Suurballe and Tarjan (1984).

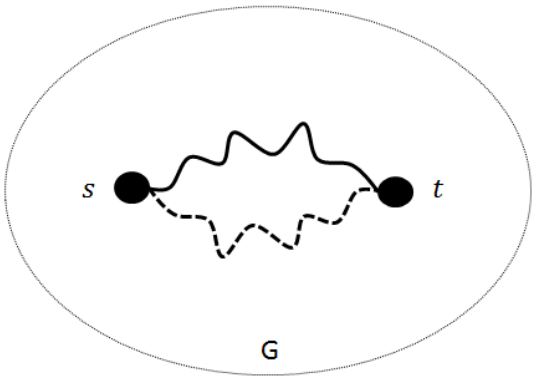


Figure 4: Edge disjoint shortest paths between a pair of vertices (s, t) as studied by Suurballe and Tarjan (1984).

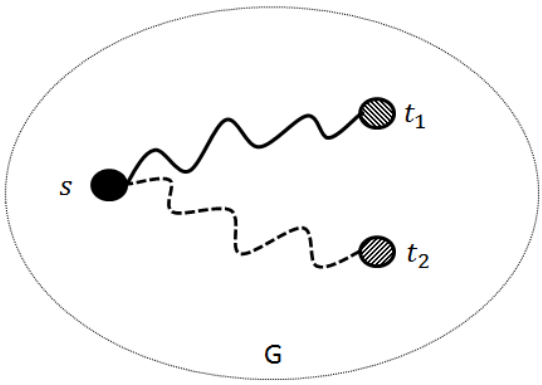


Figure 5: Edge disjoint shortest paths between a source s and two targets t_1 and t_2.

Yet another, more general, problem is finding disjoint shortest paths between two pairs of vertices: one from source s_1 to target t_1 and one from source s_2 to target t_2 (see Figure 6). The objective that is of interest for the research in this thesis is the minimal sum of the path costs. Early research by Eilam-Tzoreff (1998) on the two pairs shortest paths problem focusses, however, on two less natural objectives. Their first problem definition is: find the shortest path and a second disjoint path that is not necessarily short(est). Their second problem definition is: find two disjoint paths that are both shortest. This does not seem to be very relevant in practice, since in which case do you have a lot of shortest paths of *exactly* the same length in which you would like to find disjoint ones? Usually, in practice the shortest path will be unique or there are only a few (provided the rounding applied is not too rough). Much more practically relevant are the two objectives treated by Kobayashi and Sommer (2010): minimum sum or minimum maximal length. The former is important in this thesis and this discussion focusses

on it. It is about finding two disjoint paths of which the sum of the lengths is minimal. For the general case of an undirected graph no polynomial algorithm has been found nor has it been shown that the problem is NP-complete (Kobayashi and Sommer, 2010); also not for undirected planar graphs. Only for the special cases that the graph is undirected, planar and with sources and sinks incident to at most two faces of the graph, a polynomial algorithm is known (see Kobayashi and Sommer (2010)). In most of the instances considered in this thesis, however, this assumption of sources and sinks incident to at most two faces, does not hold. Note that the general directed version of this problem was shown to be NP-hard by Fortune et al. (1980). In Appendix A a few intuitive methods are discussed that in general fail to give a feasible and optimal solution. They give, however, good insight in this complicated problem that is still an open research area in the literature.

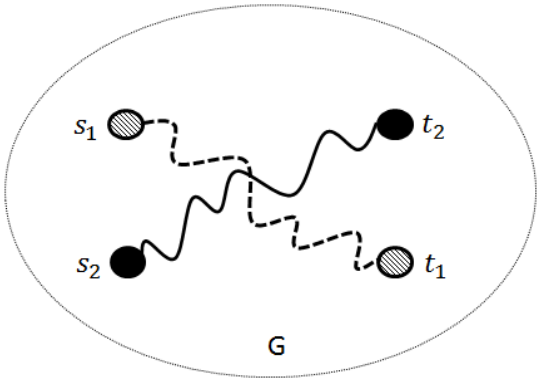


Figure 6: Edge disjoint shortest path pairs (s_1, t_1) and (s_2, t_2).

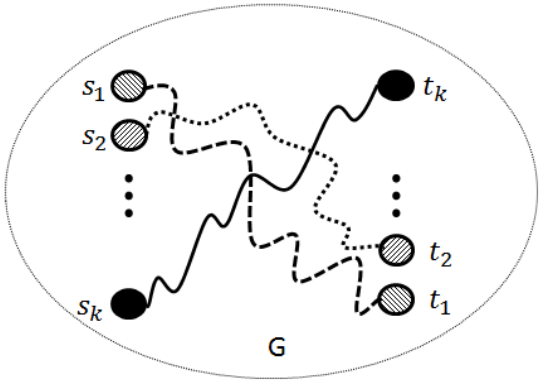


Figure 7: Edge disjoint shortest path pairs (s_1, t_1), (s_2, t_2) up to (s_k, t_k).

Even more general is the problem to find k disjoint paths for the pairs (s_1, t_1),(s_2, t_2) to (s_k, t_k), for which the sum of the path costs is minimal (see Figure 7). For $k = 2$ some special cases are known to be solvable in polynomial time. For $k = 3$ this is only the case, when all sources and sinks are incident to one face. All other special cases and $k > 3$ are still open research areas. Note that for k not fixed (so where it is still to determine how many of the pairs can be connected via a disjoint path) the problem is NP hard (Karp, 1972).

3 Mathematical formulation and prerequisites

In this section the problem sketched in Section 1.2 is described more formally and two important prerequisite algorithms for the methods developed in Sections 4 and 5 are described. The first prerequisite algorithm is Dijkstra's algorithm for finding shortest paths. It is discussed in Section 3.2. Dijkstra's algorithm is in turn used in Suurballe's algorithm, which is explained in Section 3.3. This algorithm will be used in the developed methods in Sections 4 and 5 to find edge disjoint shortest paths.

3.1 Problem formulation

The input data of the problem sketched in Section 1.2 can formally be described in the following way. Given is an undirected weighted graph $G = (V, E)$, where V denotes a set of vertices and E denotes a set of edges (ditches). Furthermore, a central office $CO \in V$ and a set of street cabinets $SC \subset V \setminus \{CO\}$ is given. Moreover, for each vertex $i \in SC$ a parameter q_i indicates the number of customers that it serves. The maximum number of customers that can be included in a ring is denoted by Cap.

To each edge $(i, j) \in E$ a length $a_{ij} > 0$ is associated. Note that $a_{ij} = 0$ if there is no edge between i and j or when i=j. This might seem strange, but it in fact ensures that a sparse representation of the length matrix is possible: length 0 means no edge and positive length means an edge. The length is, however, not the most important in this problem. More important is the cost (or weight) w_{ij} of an edge, which depends on the length of the edge and on whether on the edge already a duct is available with free space. The latter information is given in a subgraph $F = (V^F, E^F)$ of G. Contrary to G, F is unweighted and its edges are the ducts with available capacity. In other words, if $(i, j) \in E^F$ then this means that on this edge a duct with available capacity is present. Now introduce a binary parameter f_{ij} for which $f_{ij} = 1$ if $(i, j) \in E^F$, and $f_{ij} = 0$ otherwise. The weight w_{ij} of an edge $(i, j) \in E$ can then be calculated in the following way. If $f_{ij} = 0$, then $w_{ij} = a_{ij} \cdot \alpha$, where α is length unit cost of the fiber (including the digging, working, etc.). If $f_{ij} = 1$, then $w_{ij} = a_{ij} \cdot \alpha \cdot \beta$, where β is the cost factor reduction as a result of being able to use an existing pipe with available capacity. Note again that because of sparsity reasons, $w_{ij} = 0$ if there is no edge between i and j or when i=j.

Denote by R_r^C a vector of ordered street cabinets that are in ring (circuit) r and by R_r^P the same ring in terms of edges and vertices of the original graph G. Then the problem that needs to be solved, can be formulated in the following way: find rings $R_1^C, ..., R_k^C$ (and of course $R_1^P, ..., R_k^P$) in the graph G for which: $\sum_{r=1}^{k} \sum_{(i,j) \in R_r^P} w_{ij}$ is minimal, while satisfying the following constraints:

1. $CO \in R_r^C$ for all $r \in \{1, ..., k\}$ (the CO is included in each ring).
2. For all $i \in SC$, there exists an $r \in \{1, ..., k\}$ s.t. $i \in R_r^C$ (each SC is included in a ring).
3. $\sum_{i \in SC \cap R_r^C} q_i \leq Cap$ for all $r \in \{1, ..., k\}$ (maximum number of customers in ring not exceeded).

4. Each $(i, j) \in E$ is in R_r^P at most once for all $r \in \{1, ..., k\}$ (a ring is edge disjoint).

This paragraph concludes with a few remarks. First, note that the problem can also be formulated in terms of paths instead of circuits. The rings (circuits) $R_1^P, ..., R_k^P$ can also be represented by $P_1, ..., P_k$ from the CO to an imaginary copy of the CO. In both cases also a vector R_r^C of the street cabinets for which the customers are served from ring r, needs to be stored.

Secondly, note that the vertex of a street cabinet $i \in SC$ can be in multiple rings. Only one of the rings will, however, serve the customers of the street cabinet. For this ring r it holds that street cabinet i is in R_r^C.

3.2 Shortest path algorithms

An important part of the efficiency of the methods developed in this thesis stems from the chosen implementation of the shortest path algorithm. The practical case that needs to be solved is a very large graph with many nodes and edges. Moreover, shortest paths need to be computed many times to determine disjoint shortest paths. The most famous known shortest path algorithms are due to Dijkstra (1959) and Bellman-Ford (Bellman, 1958; Ford, 1956). Both are polynomial time algorithms; a straightforward implementation of Dijkstra (1959) has asymptotical complexity $\mathcal{O}(|V|^2)$, whereas Bellman-Ford has complexity $\mathcal{O}(|V||E|)$. Since in a connected graph $|E| \geq \mathcal{O}(|V|)$, it is preferred to use Dijkstra over Bellman-Ford whenever possible. Dijkstra is only able to handle non-negative edge weights, whereas Bellman-Ford is able to handle also negative edge weights and can detect negative cycles. In the practical application under consideration here no negative edge weights are present, so Dijkstra's algorithm can be used.

Next to the cases non-negative versus negative edge weights, a distinction can be made between a single source shortest path problem and an all-pair shortest path problem. In the former, the shortest path is computed from a single source to all other vertices, whereas in the latter, a shortest path is computed between all vertex pairs. In this thesis a shortest path needs to be computed between all-pairs of a relatively small subset of all vertices. Therefore, it is best to use a single source shortest path algorithm and apply it to every vertex in the subset. The most efficient algorithm known for the all-pairs shortest path problem (in terms of asymptotic complexity) in case of non-negative edge weights is simply applying Dijkstra $|V|$ times (Cormen et al., 2009).

In Algorithm 1 Dijkstra's algorithm is outlined in pseudo code (Cormen et al., 2009). The algorithm starts with various initializations in lines 1-7. The shortest path cost $d[v]$ is set to ∞ (except for the source s where it is set to 0). The predecessor $\pi[v]$ for each vertex V is set to NIL. Additionally, two sets of vertices are initialized: the set S of vertices for which the final shortest path is already determined, and the min-priority queue Q of vertices that still need to be considered. Then repeatedly a vertex from Q is selected with minimum priority (line 9), i.e. among the vertices in Q the vertex u is selected with lowest $d[u]$. Then all edges leaving this vertex u are 'relaxed' (lines 11-16). This relaxation means that if an edge leaving the vertex u to another vertex v lowers the current shortest path cost from s to u, the shortest path is adapted to use this edge.

Algorithm 1 Dijkstra(G,w,s) (Cormen et al., 2009)

Input: A weighted directed graph $G = (V, A)$ with weights $w_{uv} \geq 0$ for each arc $(u, v) \in A$, a source vertex s.

Output: the cost $\delta(s, v)$ of the shortest path from source s to each $v \in V$ and a predecessor $\pi[v]$ in the shortest path for each $v \in V$.

1: **for** each vertex $v \in V$ **do**
2: $d[v] := \infty$ {initialize shortest path cost for vertex v}
3: $\pi[v] := NIL$ {initialize predecessor for vertex v}
4: **end for**
5: $d[s] := 0$ {shortest path from s to itself is 0}
6: $S := \emptyset$ {Initialize the set of processed vertices}
7: $Q := V$ {Initialize the set of vertices still to be processed}
8: **while** $Q \neq \emptyset$ **do**
9: $u := EXTRACTMIN(\text{Q})$ {Start processing the vertex u in Q with lowest $d[u]$}
10: $S := S \cup \{u\}$
11: **for** each vertex $v \in Adj[u]$ **do**
12: **if** $d[v] > d[u] + w(u, v)$ **then**
13: $d[v] := d[u] + w[u, v]$ {relaxation operation}
14: $\pi[v] := u$
15: **end if**
16: **end for**
17: **end while**

As soon as a vertex u is extracted from Q, the shortest path cost $d[u]$ converged to the cost of the shortest path $\delta[s, u]$. As a consequence, when the algorithm finishes all shortest paths $\delta[s, v]$ are found (for a formal proof see e.g. Cormen et al. (2009)). Using the predecessor relationships, a shortest path from s to v can be constructed backwardly starting from v.

As one may notice from the discussion above, Dijkstra's algorithm will compute more shortest paths than are actually needed. In fact, from a single source only shortest paths need to be computed to a relatively small subset of all vertices. In terms of asymptotical complexity, no better algorithm is known that is faster in the worst case for this case, even not when you are only looking for a shortest path between one pair of vertices (Cormen et al., 2009). Some methods are known to speed the algorithm up in the 'average' case in the latter case, e.g. a completion check, bidirectional search or goal directed search (Holzer et al., 2005). The simplest method, the completion check, was implemented for the more general case of having a (relatively) small subset of the vertices as destinations instead of a single destination. A check can be build in that finishes the algorithm as soon as all the vertices in the subset of street cabinets are extracted in line 9 of Algorithm 1. As explained, the (optimal) shortest paths to the vertices in the subset are then found. Completing the algorithm to find the shortest paths to all remaining not yet extracted vertices is for the purposes in this thesis not necessary. Unfortunately, implementing the completion check only resulted in a faster computation when the subset of street cabinets contained very few street cabinets and they were all quite close to each other. In fact, checking in every iteration the condition whether an extracted vertex is in the array of street cabinets and checking whether the algorithm needs to continue, took more time than was saved for reasonable problem instances. To reduce the computation time of large instances somewhat, an alternative way to reduce the computation time is to ignore temporarily part of the total graph. This is implemented, but in a very conservative manner to minimize the risk of not

getting the shortest possible paths. Ignoring part of the graph can be done based on Euclidean distances or based on cost. The former is implemented in this thesis. Due to the conservative implementation only for large instances the computation time is reduced.

It was mentioned that a straightforward implementation of Dijkstra has asymptotic complexity $\mathcal{O}(|V|^2)$. Such a straightforward implementation is for instance storing the $d[v]$ in array of size 1 by $|V|$. As a result, each time when the element with the minimum key has to be extracted, the whole array has to be searched through (in the worst case). A smarter idea is to implement the min-priority queue in line 9 using a binary heap, i.e. a nearly complete binary tree. In this way Dijkstra's asymptotical complexity can be improved for sufficiently sparse ($|E| = o(|V|^2/log|V|)$) graphs (Cormen et al. (2009)). A min-priority queue is a data structure for maintaining a set S of elements that each has an associated value (called key). On this queue several operations can be executed such as insertion, extraction of the element with the minimum key and increasing a key of an element. Using binary heap, no longer an entire array has to be searched through and this results in an asymptotical complexity of $\mathcal{O}((|E|+|V|)log|V|)$ for Dijkstra. In the programming in this thesis an implementation of Dijkstra with binary heap was used that is available online (Gleich, 2009). Note that the asymptotical complexity can even be further improved by using Fibonacci heap to $\mathcal{O}(|V|log|V| + E)$. Fibonacci heap uses a collection of trees, instead of just a single binary tree. According to Cormen et al. (2009) Fibonacci heap is, however, less desirable for most practical applications because it is quite complicated to program.

3.3 Finding two edge disjoint shortest paths

In Section 2.2 an overview of the literature on the edge disjoint shortest paths problem was presented. Below two different methods are discussed that are able to find two disjoint shortest paths from a single source vertex to two destination vertices t_1 and t_2 (see Figure 5). This problem reduces in the following way to the problem that searches two disjoint paths between a pair of vertices for which the sum of the path costs is minimal: introduce an artificial sink vertex t' that is connected to the two destination vertices t_1 and t_2. Intuitively, the edges (t_1, t') and (t_2, t') should ideally be given a cost of 0, but this is not possible since a sparse matrix representation of the weight matrix is used where a weight of 0 means that there is no edge. This is solved by giving these edges a small positive weight. The artificial sink t' reduces the problem to finding two disjoint paths between a pair of vertices (for which the sum of the path costs is minimal). For this problem there exist efficient algorithms: one due to Suurballe and Tarjan (1984) and one due to Bhandari (1999). The algorithm by Suurballe and Tarjan (1984) is implemented in this thesis and is discussed in more detail below. Afterwards, some remarks are made on the algorithm by Bhandari (1999) and the choice for the algorithm by Suurballe and Tarjan (1984) is motivated.

3.3.1 Suurballe's algorithm

Suurballe (1974) and Suurballe and Tarjan (1984) developed an algorithm for the problem with the objective to minimize the sum of the costs of two disjoint paths between a source vertex s and a target vertex t. Suurballe (1974) dealt with vertex disjoint shortest paths. The ideas where extended to the edge disjoint shortest paths in Suurballe and Tarjan (1984). In the sequel, the

algorithm will simply be called "Suurballe's algorithm", as is common practice in the literature. The algorithm has the same asymptotical complexity as Dijkstra: $\mathcal{O}((|E|+|V|)log|V|)$ (using binary heap). As discussed in Section 3.2, this can be improved to $\mathcal{O}(|V|log|V|+E)$, at the cost of high programming effort. In the Matlab code of this thesis a binary heap implementation of Dijkstra programmed by Gleich (2009) is used.

The algorithm consists of six steps which are outlined in more detail in Algorithm 2 (Bhandari, 1999; Suurballe, 1974; Suurballe and Tarjan, 1984).

Algorithm 2 Suurballe(G,w,s,t)

Input: A weighted directed graph $G = (V, A)$ with weights $w_{uv} > 0$ for each arc $(u, v) \in A$, a source vertex s and a target vertex t.

Output: the cost of two disjoint shortest paths P_1 and P_2 from source s to target t and a representation of the path P_1 and the path P_2.

1: Run Dijkstra(G,w,s) to obtain $\delta(s, v)$ and $\pi[v]$ $\forall v \in V$. Construct the path $\overline{P_1}$ from s to t using $\pi[v]$. {see Algorithm 1}

2: Transform the graph G to the graph G' by applying the following transformation on the weights: $w'(i, j) = w(i, j) + \delta(s, i) - \delta(s, j)$ $\forall(i, j) \in A$.

3: Remove in G' the arcs in the opposite direction of the shortest path $\overline{P_1}$ found in step 1. Then switch the direction of the arcs in $\overline{P_1}$ in the graph G'.

4: Run Dijkstra(G',w',s) to get $\delta'(s, v)$ and $\pi'[v]$ $\forall v \in V$. Construct the path $\overline{P_2}$ from s to t using $\pi'[v]$. {see Algorithm 1}

5: Create a graph G^* consisting of the arcs of the paths from s to t found in step 1 and 4. All 'reversed arcs' used in the path found in step 4, are removed from the graph together with their counterparts in the other direction.

6: Find the disjoint paths P_1 and P_2 from s to t in the following way: run Dijkstra(G^*,w^*,s) to find P_1 with corresponding cost. Remove the arcs of P_1 from G^* to form the graph G^{**} and again run Dijkstra(G^{**},w^{**},s) to find P_2 with corresponding cost.

In step 1 of the algorithm a shortest path tree is constructed from source s. This tree is used in step 2 to adapt the weights of the graph. The reason the weights are transformed is the following. Intuitively one would like to find the first path and then remove the arcs of this path from the graph. Their counterparts in the other direction are changed (or created if it was not present) to minus the cost of the arc used in the path. Afterwards, a path can be determined in this new graph and steps 5 and 6 remain unchanged. This approach is, however, not a very clever idea, since then a slower algorithm that is able to handle negative weights has to be used instead of Dijkstra. Suurballe's algorithm avoids having to use another shortest path algorithm by transforming the weights in such a way that they remain non-negative and that still guarantee finding the optimal shortest path. The latter can be observed in the following way (Bhandari (1999)): consider a path $P = sv_1v_2...v_nt$ from source s to target t. The transformed length of this path is $\delta'(s,t) = w'(s,v_1) + w'(v_1,v_2) + ... + w'(v_n,t) = w(s,v_1) + \delta(s,s) - \delta(s,v_1) + w(v_1,v_2) + \delta(s,v_1) - \delta(s,v_2) + ... + w(v_n,t) + \delta(s,v_n) - \delta(s,t) = \sum_{(i,j)\in P} w(i,j) - \delta(s,t)$, so the length of an arbitrary path from the source s to the sink t in G' is less than the cost of the path in G by a constant equal to the length of the first shortest path from s to t. In this way, the ranking of the paths (in terms of their length) remains unchanged under the transformation and finding the optimal shortest path can be guaranteed.
In step 3 the arcs from the shortest path are reversed; an idea often used in flow problems. If

the shortest path in this 'residual graph' (found in step 4) uses some of these arcs, then they are not in the final paths anymore; they are 'neutralized'.
Subsequently, in step 5 a graph is formed of all arcs used in the two paths, while deleting the arcs that 'neutralized' each other. In step 6 the two disjoint paths are found for which the sum of the path costs is minimal. All arcs in the graph are going to be used in one of the two disjoint paths. The easiest way to determine the exact paths and their cost is by running Dijkstra to find the first path, deleting the arcs, and running Dijkstra for a second time. As one can imagine, the graph in step 6 is extremely straightforward. Using Dijkstra is just a very practical way to obtain the paths, since it allows reuse of code and does take negligible computation time. Other methods to determine the final paths are, of course, also possible.

3.3.2 Motivation for the choice of Suurballe's algorithm over Bhandari's algorithm

To motivate the choice for Suurballe's algorithm, first a short description of Bhandari's algorithm is needed. In the previous paragraph the main difference between the two algorithms has already been pointed out: Suurballe uses a transformation to keep all weights of the graph non-negative, whereas Bhandari does not. This makes Suurballe's algorithm less intuitive than Bhandari's algorithm, but Bhandari has to use another shortest path algorithm that can handle negative weights. One could use the 'slow' Bellman-Ford algorithm for this, but Bhandari (1999) comes up with an alternative, which he names the 'modified Dijkstra algorithm'. Recall that the Dijkstra algorithm (see Algorithm 1 in Section 3.2) labels in each iteration one vertex u permanently, meaning that it is checked whether the edges to the vertices adjacent to u can be relaxed. More importantly, when extracting a vertex u from the set Q in line 9 of Algorithm 1, the shortest path from the source s to vertex u is found. In the 'modified Dijkstra algorithm', vertices are still extracted, edges are relaxed, but they are no longer necessarily 'permanently' labeled. In other words, as a result of the relaxing, the shortest path of a vertex that is no longer in Q can still decrease by relaxing a negative weighted edge. If this happens, it needs to be reinserted in Q to process it again, which is the main idea of the 'modified Dijkstra algorithm'. This reinserting implies that the algorithm no longer has the same asymptotical complexity as Dijkstra, but in the worst case processes $|V| * (|\overline{P_1}| - 1)$ extra vertices. $|\overline{P_1}|$ denotes here the number of vertices in the first shortest path found in step 1 of the algorithm. In practice the computation time will probably not increase that much. Bhandari (1999) advocated the use of this algorithm over Suurballe's algorithm mainly because of the time it takes in Suurballe's algorithm to recompute all the weights. Testing in Matlab showed, however, that this can be done very fast. Most of the computation time stems from the two calls of Dijkstra in steps 1 and 4 of Algorithm 2. The modified Dijkstra algorithm will only make these two calls significantly slower, resulting in a larger computation time. It might be that in other programming languages the transformation on the weights could take long, but at least in Matlab this is not the case, since it is very efficient in handling matrices. This is the main reason for the choice not to implement Bhandari (1999) and to use Suurballe's algorithm.

3.3.3 Simple methods for the two pairs disjoint shortest path problem

In Section 2.2 it was identified that the problem of finding two disjoint paths, one from source s_1 to target t_1 and one from source s_2 to target t_2, for which the sum of the path costs is minimal, is an open research area for undirected planar graphs. In Appendix A two simple

approaches are shown to fail in general. Perhaps, the easiest attempt is to use an approach which will be referred to as the 'deletion method'. This approach works as follows: search for the shortest path between s_1 and t_1, delete this path from the graph and subsequently search for the shortest path from s_2 to t_2 in this new graph. Then repeat this procedure a second time, but in a reversed order: first search the shortest path from s_2 to t_2, delete this path from the graph and search for the shortest path from s_1 to t_1 in the new graph. From these two options the cheapest (feasible) option is chosen. This approach has two problems: firstly, it may not find a feasible solution, whereas there actually is one. Secondly, it may not find the optimal solution. These two problems are illustrated with examples in Appendix A. Another approach would be to try to use Bhandari (1999) in the more general case of two pairs of vertices. In some graphs this approach is successful, but in many instances it fails. Examples are again given in Appendix A.

3.3.4 Extension of Suurballe's algorithm for pinched circuits

In practice, the edge disjointness restriction does not always have to be obeyed completely. Close to a street cabinet it is e.g. often allowed to use a ditch in two directions for the same ring. This is called a pinched circuit. This extension of the problem was not implemented in Matlab, but in this section it is outlined how pinched circuits can be added to Suurballe's algorithm. Given is a maximal length of a pinched circuit, given by the parameter *pcl*. The extension is illustrated by the simple example in Figure 8. The cost of each edge is indicated on each edge. Moreover, the length of each edge is given. These are not illustrated in the Figure, but assume that EGt (or Gt, of course) can become a pinched circuit, since these edges are relatively short. All other candidate edges (sA,sB and Ft) are on itself already larger than *pcl*. The goal is to find two disjoint shortest paths between source s and target t for which the sum of the path costs is minimal, while allowing pinched circuits of length *pcl*. The trick to incorporate this possibility of a pinched circuit is as follows: introduce two artificial nodes E' and G'. Moreover, introduce the edges EE', $E't$, GG' and $G't$ with costs: $c_{EE'} = c_{E't} = \frac{1}{2}(c_{EG} + c_{Gt}) = 2$ and $c_{GG'} = c_{G't} = \frac{1}{2}c_{Gt} = 1$. The new graph is illustrated in Figure 9. In this new graph Suurballe's algorithm can be applied to find the two shortest disjoint paths, while allowing pinched circuit. To summarize: for each node in a potentially allowed pinched circuit, a new artificial node needs to be introduced. Each artificial node needs to be connected to the endpoint of the pinched circuit and its original node. The sum of the two new edges must be set equal to the shortest path cost to the endpoint of the pinched circuit.

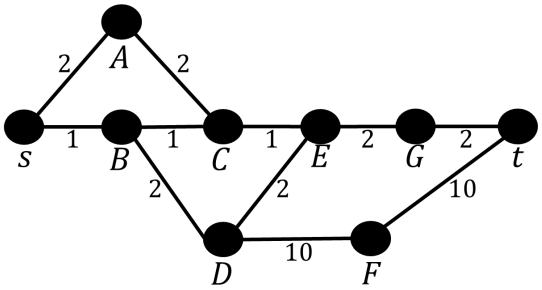


Figure 8: Example to illustrate how to allow pinched circuits in Suurballe's algorithm.

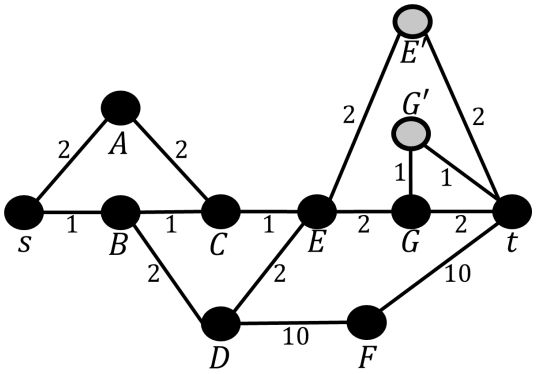


Figure 9: Graph after applying the trick (introducing nodes E' and G').

4 Initial solution

In this section a method is outlined to find an initial solution of good quality. It is important that the solution is of good quality, since due to the edge disjointness restriction, local search to improve the solution takes quite long for large instances (see Section 7.6). Firstly, different VRP heuristics to construct initial solutions are discussed in Section 4.1. Their advantages and disadvantages are mentioned and in particular their suitability to the problem studied in this thesis is analysed. Secondly, in Section 4.2 the algorithm to construct an initial solution for the problem in this thesis is given. The algorithm that was developed in earlier research by TNO is presented in Section 4.3.

4.1 VRP heuristics to construct an initial solution

For some combinatorial problems it is fairly easy to construct an initial feasible solution (e.g. randomly). For the problem studied here, it is not so straightforward at all. This is mainly due to the restriction that each ditch edge can only be used once in a ring. From the Vehicle Routing Problem (VRP) literature, four main techniques can be distinguished that allow construction of an initial feasible solution:

1. the savings algorithm of Clarke and Wright (1964);
2. an insertion heuristic;
3. cluster first - route second;
4. route first - cluster second.

In the next paragraphs these are dealt with shortly and their suitability for the problem is investigated.

4.1.1 Clarke and Wright

The simplest known VRP heuristic is the savings algorithm of Clarke and Wright (1964). This heuristic initially assigns each customer directly to the depot. Then an ordered list of savings for each pair of customers is composed. The savings are the cost reductions resulting from merging the routes of two customers into one. Then the list of savings is run through and mergers of routes are performed if they are feasible. Typical to this approach is that a strong geographical clustering arises. A disadvantage could be that there are more routes constructed than that there are vehicles available. The algorithm of Clarke and Wright could be used in two distinctive ways for the problem in this thesis. A straightforward application of the algorithm of Clarke and Wright produces a solution that is not necessarily edge disjoint. This solution could be repaired to make it edge disjoint, but it is very likely that this will lead to high costs; especially for the instances studied here that have very cheap paths available in the graph (existing ducts). These instances are illustrated in more detail in Section 6. Another possibility is to try to adapt the algorithm of Clarke and Wright to make sure that edge disjoint rings are constructed from the start. This would mean that the algorithm starts by constructing two edge disjoint paths between the central office and each street cabinet. However, then in the

next step a problem arises: merging routes may result in edges that are used more than once. On the other hand, only considering a merger if both routes are completely disjoint seems too restricting. It is possible to consider each merger and then recompute disjoint paths in such a way that the merger can be done in a disjoint way, but this has major disadvantages:

1. the disjoint shortest paths cannot be calculated optimally; the deletion method has to be used.
2. it involves computing shortest paths very often, leading to a large computation time.

Using the deletion method in early stages of the construction of a solution may result in very bad solutions, since optimal paths will be replaced by more expensive paths early on, while actually in the end solution this might not have been necessary. In that light, it is expected that this will produce worse solutions than using the standard Clarke and Wright algorithm and afterwards correct the solution by the deletion method. The latter is also faster, so integrating the disjointness restriction into the algorithm of Clarke and Wright is not a good idea. Not taking the edge disjointness restriction into account directly while using the algorithm of Clarke and Wright is an option, but since the objective of the research in thesis is to come up with methods that integrate the decisions of clustering and edge disjoint routing more than in the cluster first - route second heuristic investigated by TNO (i.a. Phillipson (2013b) and Phillipson (2013c)), no further effort is spent to investigate this further.

4.1.2 Insertion heuristic

A more promising approach than the algorithm of Clarke and Wright for the problem studied here is the use of an insertion heuristic. Insertion heuristics construct a feasible solution by repeatedly and greedily inserting an as of yet unrouted customer into a partially constructed feasible solution (Campbell and Savelsbergh, 2004). Advantages of insertion heuristics are that they are fast, produce decent solutions, are easy to implement and can easily be extended to handle complicating constraints. All these advantages are important to the problem in this thesis. For example, as described in the problem definition in Section 1.2, the algorithm needs to be very fast. Moreover, the use of an existing ditching pattern and the edge disjointness of a ring are very complicating side constraints, which an insertion heuristic might be better able to handle than other approaches. Finally, it is recognized by Desrosiers et al. (1995) that insertion heuristics are typically used to construct an initial feasible solution in local search and metaheuristics for vehicle routing and scheduling problems.

There are two different approaches that can be taken in an insertion heuristic: a sequential or a parallel approach. The former builds routes one at a time, whereas the latter builds those all at the same time. In other words, the sequential approach works as follows (Potvin and Rousseau, 1993): a route is initialized with a 'seed' customer, and the remaining unrouted customers are added until the constraints do not allow it anymore. Then a new route is initialized and the procedure is repeated until all customers are included in a route. The 'seed' customer can, for instance, be chosen by taking the farthest customer from the depot. Which unrouted customer to add is, for example, decided by first computing for each customer the best feasible insertion place in the route, and then selecting the one out of these that has the lowest insertion cost (Solomon, 1987). Potvin and Rousseau (1993) come up with the parallel approach mainly to avoid the important disadvantage of the sequential approach that the last routes are of a very poor quality, since the last remaining customers are usually spread over the geographical area.

Their approach is as follows: estimate the number of routes needed and then initialize all of them taking a seed from each route obtained by the sequential algorithm of Solomon (1987). In this way each route is assigned one customer. For the rest of the unrouted customers the best feasible insertion place is computed for each of the initial routes. Then using a regret function it is decided which customer is routed first and in which position in which route. This regret function measures how urgent it is to include a customer into a route and how many relatively cheap alternative routes there are still available for it. To avoid the mentioned disadvantages a parallel approach is used to come up with an initial solution for the problem in this thesis.

4.1.3 Cluster first - route second

The original approach taken by TNO and the research by Phillipson (2013b) is to first cluster the street cabinets and afterwards route. In the clustering phase no attention at all is paid to a ditching pattern or cheaply available pipes; the SC's are clustered purely based on mutual Euclidian distances. This approach might, however, seriously harm the solution quality, since very expensive detours might be necessary to repair a solution afterwards. It is possible to make the cluster decision based on costs taking into account a ditching pattern and cheap available pipes. Then, however, problems arise when a network of available pipes is available (as is often the case in the practical application studied here). A lot of pipes will be doubly used and when the routing decision is subsequently considered again large additional costs might be incurred. The cluster first - route second heuristic used by TNO is described in more detail in Section 4.3.

4.1.4 Route first - cluster second

Another possibility is to first construct one big tour and then cut that tour into pieces according to the capacities of the rings. In this problem this approach has, however, several disadvantages. Firstly, it is hard to find one big tour that is completely edge disjoint. The restriction in this thesis does not require that. It only prescribes edge disjointness in every ring. Not taking into account the edge disjointness restriction into the routing initially is possible, but not suitable for this thesis. The goal of the thesis is namely to create a methodology that does take the ditching pattern and available pipes into account in an early stage with the intent to create a coherent approach.

4.2 The insertion heuristic

In this section the method to find an initial solution of good quality for the problem defined in Section 3.1 is presented. Its pseudo-code is given in Algorithm 3. All the references to lines in the section below are to this algorithm. Some technical details are not included in this pseudo-code to improve the readability. The output of the heuristic are k rings that are all edge disjoint, but not mutually edge disjoint (between the rings). The objective is to minimize the sum of the ring costs. The algorithm starts with initializing the k rings by inserting one SC and the CO in each of them (k seeds need to be selected). Here k is a constant such that all street cabinets can in fact be included in a ring, while the restriction of the maximum capacity

of a ring is obeyed (see line 2). At the end of this section this elaborated further. To be able to start with a good initialization, first all the shortest paths between the SCs are calculated using Dijkstra on the weight matrix W (see line 1). Then the initialization can start (lines 4-10) for which the following seed selection methods were studied:

1. *Random*: the most simple way to initialize all rings is to assign a street cabinet randomly to each ring. This street cabinet is then connected to the central office by two disjoint paths that are calculated by Suurballe's algorithm. It is, however, very likely that this is not a really auspicious attempt.

2. *Centroid with or without CO*: to obtain 'seeds' to initialize the rings, it might be an interesting idea to use a clustering algorithm first and pick a seed from each cluster. Phillipson (2013b) developed a clustering algorithm to group the street cabinets in such a way that the sum of the Euclidean distances of each street cabinet to the centroid of the cluster it is assigned to, is minimal. The central office can be forced to be present in each cluster or can be left out of the analysis. After clusters have been formed, the street cabinet closest to the centroid of each cluster is chosen as a seed. In this way, both the capacities of the rings as well as the geographical spread is taken into account.

3. *Most spread*: this assigns seeds to the rings in a iterative procedure. For ring R_1^C add the CO and the SC furthest away (in terms of the shortest paths just computed). For ring R_2^C add the CO and the SC furthest away from the SC in ring R_1^C and the CO. For ring R_3^C add the CO and the SC furthest from R_1^C and R_2^C, etc. This is continued until all k rings contain the CO and one SC. In formula this is given by the expression in line 5 of Algorithm 3. This initialization (that is also used in Naji-Azimi et al. (2010)) ensures that the seeds are somewhat spread over the region with respect to the real costs in the graph (not necessarily geographically according to Euclidean distances). The testing in Section 7.1 showed that this method to choose the seeds performed better than the other two approaches for the most interesting instances.

After the seed street cabinets are chosen, the actual rings can be built. In other words, for each of the rings $R_1^P, ..., R_k^P$, the actual disjoint paths (CO to SC and then back to CO) are calculated using Suurballe's algorithm (see lines 4-10).

Then each ring is initialized and the remaining SCs (see lines 11 and 12) should be included in the different rings in such a way that the total cost of the rings is as low as possible. This is done in the following way: insert the remaining SCs one by one; each time looking at the cheapest ring and position to insert it. The order in which the SCs need to be inserted can be determined in several ways. Three possibilities are investigated in this thesis:

1. *Random*: the SCs are ordered in a random way and are inserted in this order.

2. *Disjoint insertion cost*: each time when deciding which SC to insert next, the costs is calculated to insert each unconnected SC in each ring in such a way that the edge disjointness constraint is satisfied. The simplest way to choose which SC to insert is to pick the one which can be inserted cheapest in a ring. This option was implemented. A more sophisticated idea would be to decide which SC to insert next based on some kind of regret function that decides which SC is most urgent to insert in a ring. Choosing just the cheapest can lead to unwanted situations. A simple example is the following: consider

two SCs, l_1 and l_2, and two rings, r_1 and r_2. Ring r_1 is almost full, and only has space left for one of the SCs. Now assume that SC l_1 has insertion costs 0 and 1 for rings r_1 and r_2, respectively. SC l_2 has insertion costs 1 and 5, respectively. According to the simple approach SC l_1 will now first be inserted into ring r_1. It is, however, immediately clear that it was better to insert l_2 in this ring. Such situations may occur and are, of course, not desirable and this effect can be diminished to a certain extent by using this alternative objective. Unfortunately, testing showed that this implementation with disjoint insertion costs takes too much time, since disjoint paths need to be computed too often to keep the computation time within the desired limits that are formulated in the problem definition of this thesis. Therefore, no further investigation was started to improve upon the simple criterion of lowest cost by some more sophisticated approach. This can be an interesting subject for further research to gain better solutions for situations where computation time is not so much of an issue.

3. *Non-disjoint insertion cost*: each time the unconnected SC with the lowest non-disjoint insertion cost is inserted. These non-disjoint insertion costs are directly calculated from the shortest path costs that are calculated in line 1 of Algorithm 3. The edge disjointness restriction is not necessarily satisfied and this option is therefore a compromise with regard to the previous option. It is less reliable, since non-disjoint cost can be quite deceiving, because of the inexpensive paths in the graphs that are studied. It gives, however, some indication and is therefore likely to perform better than a random order. Recall that the insertion of a SC l between i and j involves two new paths: from i to l and from l to j (see Figure 10). Moreover, the path from i to j is removed from the ring. The sum of the new path costs minus the old path cost define the non-disjoint insertion costs. For the SCs that remain to be inserted the minimum non-disjoint insertion cost over the different rings and positions is determined. The SC with the lowest minimum is inserted first.

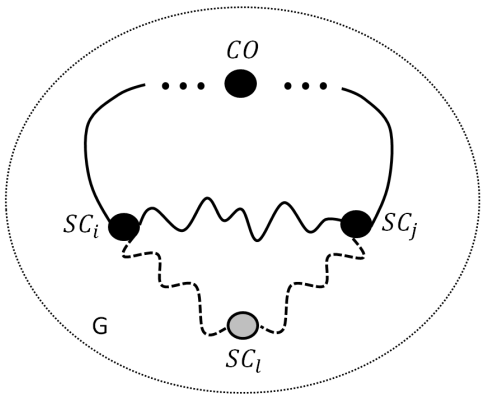


Figure 10: Illustration of the insertion of SC l in between SC i and SC j

To decide which ring and position is actually cheapest for the insertion of a certain SC, for each ring and position the (disjoint) cost is computed (see lines 13-21). An insertion of SC l in a ring r between SC i and SC j is illustrated in Figure 10. To compute the cost of an insertion the following procedure is used: remove the edges of the path from SC i to j from the ring. Then create a subgraph where the edges of this (disrupted) ring are deleted from the original graph. In this subgraph two disjoint paths are sought (from SC l to SC i and from SC l to SC j) in such a way that the total sum of the path costs is minimal. This is done using Suurballe's algorithm (see lines 16-18) that is explained in more detail in Section 3.3. Removing the current ring from the graph that is used to search the new paths to connect SC l with the ring, ensures that no

edges are used twice in a ring and the ring remains edge disjoint. Note that the insertion using Suurballe's algorithm ensures an optimal insertion, but fixing the rest of the ring makes the ring in the end not necessarily optimally routed. This is, however, the best that can be achieved with currently known disjoint shortest path algorithms (Section 2.2).

Given that SC l needs to be inserted next, the following steps are executed. Firstly, for each ring where adding SC l does not lead to exceeding the capacity, the non-disjoint insertion cost is computed for each position. Then a list is composed of all the just computed costs from low to high. Each element in the list corresponds to a ring and a position in that ring. Starting from the top of the list, the disjoint insertion costs are being computed. If the considered insertion is cheaper than the one found until that moment, the cheapest possibility is updated. In this way, an attempt is made to quickly find good disjoint solutions. Each time before considering the next option in the list, the non-disjoint cost is checked against the lowest disjoint insertion cost known until that moment. When the former is higher, there is no need to compute the disjoint shortest path costs anymore. In this way, a lot of computation time is saved. Another computation time saving technique is that before actually using Suurballe's algorithm, first the shortest paths (calculated in line 1) are constructed from the predecessor relations to see if they are already coincidentally disjoint. If so, there is no need to use Suurballe's algorithm and computation time is saved.

Finally, some remarks need to be made about k, the initial number of rings that is chosen. Each street cabinet serves a fixed number of customers. In practice often different bins are used and not the actual number of customers. These bins are less sensitive to (small) changes over time, making the results better useful in practice. A special case is where only one bin is defined: each street cabinet carries equal weight. Then the capacity restriction of the rings reduces to a restriction on the number of street cabinets in a ring. The lower bound on the number of rings is defined as $k = \lceil \sum_{i \in SC} q_i / Cap \rceil$ (see line 2 of Algorithm 3). For this special case this lower bound is always achieved in the insertion heuristic. However, when multiple bins or a actual number of customers are used, two things can prevent achieving the lower bound in the insertion heuristic:

1. It is theoretically not possible to include the SCs in k rings.
2. It is theoretically possible to include the SCs into k rings, but the insertion order in the insertion heuristic caused that not all SCs fit in k rings.

The first situation will probably not soon occur in practical instances, but imagine the following simple example (Phillipson, 2013b): 5 street cabinets are given, all with weight 4. The rings have capacity 10. The lower bound is then two rings, but obviously three rings are needed in this example. The second situation occurs quite frequently in large instances for the insertion heuristic. Note that the cluster first - route second approach (Section 4.3) will make sure that the lower bound on the number of rings is used. But this is not necessarily an advantage, since it might be cheaper to just add a few extra rings than to make large detours as a result of sticking to the lower bound. Near the end of the insertion heuristic it can happen that there are still SCs left to insert, but none of the rings has enough available capacity left. Many rings are not entirely full (otherwise the lower bound would be incorrect), but it is just not enough to insert this SC that has many customers. For example, consider an instance with two rings that each have space left for 200 customers. One SC with 300 customers remains to be inserted. An extra ring will be needed in this case. Luckily, if it reduces the cost this extra ring can become redundant in the local search phase. Further, note that if the street cabinets with many

customers would have been inserted first in the rings, two rings would probably have sufficed.

Adding an extra ring at the end of the insertion heuristic can be quite expensive. It would have been a lot cheaper if it had been added from the start. This raises an interesting question: can the cost of the insertion heuristic be lowered (on average) if a certain percentage of extra rings is added to the lower bound from the start. E.g. between 5% or 20% extra rings. The computation time requirement (at most minutes), prohibits calculating an initial solution for multiple number initial of rings. For instances of interesting sizes, this would clearly take too long. Therefore, it is worth the effort to try to take the best possible number of rings from the start. In Section 7.3 this is investigated.

Algorithm 3 InsertionHeuristic

Input: A weighted undirected graph $G = (V, E)$ with weights $w_{ij} > 0$ for each edge $(i, j) \in E$, a set of street cabinets $SC \subset V$, and a central office $CO \in V \setminus SC$. A parameter q_i for all $i \in SC$ indicating the number of customers it serves. A parameter Cap indicating the maximum number of customers in a ring.

Output: k rings $R_1^P, ..., R_k^P$ (and the $R_1^C, ..., R_k^C$), that are edge disjoint (not mutually) and their corresponding costs $cr_1, ..., cr_k$.

1: Compute the shortest path between all SCs using Dijkstra(G,w,s) for all $s \in SC$ (see Algorithm 1 in Section 3.2). Save the cost in a matrix C of size $|SC|$ by $|SC|$.
2: $k := \lceil \sum_{i \in SC} q_i / Cap \rceil$ {A lower bound on the # of needed rings}
3: $R_0^C := \{CO\}$
4: **for** $r = 1, ..., k$ **do**
5: $i := \text{argmax}_{i \in SC}\ C(\bigcup_{t=0}^{r-1} R_t^C, i)$. {"Most spread" or use one of the alternatives (see page 24).}
6: $R_r^C := \{CO, i\}$.
7: Run Suurballe(G,w,CO,i) to get P_1 and P_2 and the corresponding cost cp_1 and cp_2 (see Algorithm 2 in Section 3.3).
8: Initialize R_r^P by combining the paths P_1 and P_2.
9: $cr_r := cp_1 + cp_2$.
10: **end for**
11: **while** $SC \setminus \{R_1^C \cup ... \cup R_k^C\} \neq \emptyset$ **do**
12: Select an $l \in SC \setminus \{R_1^C \cup ... \cup R_k^C\}$ according to some priority rule.
13: **for** $r = 1, ..., k$ **do**
14: **if** $\sum_{i \in SC \cap R_r^C} q_i + q_l \leq Cap$ **then**
15: **for** $i = 1, ..., |R_r^C|$ **do**
16: Introduce a new vertex t' and define $G^* = (V^*, E^*)$ by $V^* = V \cup \{t'\}$ and $w(R_r^C(i), t') := w(R_r^C(i+1), t') := w(t', R_r^C(i)) := w(t', R_r^C(i+1)) := \epsilon > 0$.
17: Run Suurballe($G^* \setminus \{R_r^P \setminus P(i,j)\}$,$w$,$t'$,$l$), where $P(i, j)$ denotes the current path between i and j in the ring R_r^P (see Algorithm 2 in Section 3.3). If a solution exists, denote by P_1 and P_2 the two disjoint paths and by cp_1 and cp_2 the corresponding costs. If no solution exists set cp_1 and cp_2 to a high number M.
18: $cost(r, i) := cp_1 + cp_2 - c(i, j)$.
19: **end for**
20: **end if**
21: **end for**
22: Use the combination of $r \in \{1, ..., k\}$ and $i \in \{1, ..., |R_r^C|\}$ s.t. $cost(r, i)$ is minimal to adapt the rings:
23: $R_r^C := \{R_r^C(1), ..., R_r^C(i), l, R_r^C(i+1), ..., R_r^C(end)\}$
24: Insert l in R_r^P in between SC $R_r^P(i)$ and SC $R_r^P(j) := R_r^P(i+1)$ by deleting the path $P(i, j)$ and using P_1 and P_2 corresponding to the pair (r, i) instead.
25: $cr_r := cr_r - cp(i, j) + cp_1 + cp_2$, where $cp(i, j)$ is the cost of the current path from (i, j) in ring r, cp_1 the cost of path P_1 and cp_2 the cost of path P_2.
26: **end while**

4.3 Cluster first - route second heuristic

In earlier research by TNO (i.a. Phillipson (2013b,c)) a cluster first - route second approach was used, which separates the problem into two main steps:

1. Which street cabinet is served by which fiber ring?
2. How will each fiber ring run? Each ring that is constructed should be edge disjoint. In other words, it is not allowed that an edge is used twice within one ring.

The first step is elaborated in Phillipson (2013b). In that paper a method is presented for solving a constrained k-means clustering problem in a fast way.

In the second step (not published, but implemented in the tool PlanXS developed by TNO) an attempt is made to connect all cabinets that belong to the same cluster by edge disjoint paths. To do this, an initial solution is determined by solving the underlying TSP problem. This gives an initial solution that is not necessarily feasible: multiple paths in the ring might use the same edges. To try to get a feasible solution the 'deletion method' (see Section 3.3 and Appendix 2.2) is used. The whole procedure is given in more detail in Algorithm 4.

Algorithm 4 ClusterFirstRouteSecond

Input: A weighted undirected graph $G = (V, E)$ with weights $w_{ij} > 0$ for each edge $(i, j) \in E$, a set of street cabinets $SC \subset V$, and a central office $CO \in V \setminus SC$. A parameter q_i for all $i \in SC$ indicating the number of customers it serves. A parameter Cap indicating the maximum number of customers in a ring.

Output: k rings $R_1^P, ..., R_k^P$ (and the $R_1^C, ..., R_k^C$), that are edge disjoint (not mutually) and their corresponding costs $cr_1, ..., cr_k$.

1: $k := \lceil \sum_{i \in SC} q_i / Cap \rceil$. {The number of rings}
2: Use the clustering algorithm of Phillipson (2013b) to divide the SCs over clusters $C_1, ..., C_k$.
3: **for** $r = 1, ..., k$ **do**
4: Compute the shortest path between all $s \in C_r$ using Dijkstra(G,w,s) for all $s \in C_r$.
5: Solve a TSP for C_r to obtain R_r^C, the order of the SCs in ring r, the paths of the ring R_r^P, and the cost cr_r of the ring. {R_r^P might not yet satisfy the edge disjointness restriction.}
6: Create a list L which contains paths that use one or more of the same edges.
7: **while** $L \neq \emptyset$ **do**
8: Take the first conflict from the list and use the deletion method to fix the conflict. To avoid new conflicts, delete the rest of the ring from the graph.
9: **if** the deletion method gives a feasible solution **then**
10: Update R_r^P and cr_r.
11: **else**
12: STOP
13: **end if**
14: Update the list L of conflicting paths in R_r^P.
15: **end while**
16: **end for**

Note that the while-loop (lines 7-15) finishes in a finite number of steps. In fact, since each time the rest of the ring is fixed, it will finish in at most $R_r^C - 1$ steps. When the deletion method fails to find a feasible solution, the whole algorithm finishes without finding a solution (line 12). Assuming that a solution does exist, a remedy would be to take into account more than two conflicting paths at the same time, but no methods are known for this more general case.

5 Local search

5.1 Neighborhood moves

After an initial solution is formed with one of the initial solution heuristics discussed in Section 4 local search can try to improve the solution. Important to note is that in this phase of the algorithm only changes are made that do not affect the feasibility of the solution. There are several different options possible: some acting on a single ring, others acting on multiple rings simultaneously. These options are now discussed in more detail. To decrease the cost of a ring one could try to do the following neighborhood operations (see e.g. Kindervater and Savelsbergh (1997)):

1. $k-$exchanges, i.e. replace a set of k edges by another set of k edges. According to Kindervater and Savelsbergh (1997) in practical applications only 2-exchanges and 3-exchanges are relevant, since otherwise the computation time becomes too large.

2. relocation, i.e. relocating an SC to another position in the ring (or more generally relocating a set of l-consecutive nodes).

Also neighborhoods for multiple rings can be defined. In this way SCs that were assigned to a certain ring in the initial solution can be transferred to other rings. Among the possibilities are the following:

1. exchange an SC from one ring with an SC from another ring.

2. relocation, i.e. relocate an SC to another ring. It then has to be decided in which ring and in which position the SC is inserted. An SC will only be relocated to another ring if that ring has enough capacity left.

These different neighborhoods could for example be used within a metaheuristic, such as tabu search or simulated annealing. These metaheuristics are, however, only useful if savings of a certain neighborhood operation can be calculated very quickly. Unfortunately, this is not the case for the problem studied in this thesis; already for medium sized instances the computation of edge disjoint paths becomes too time intensive to do them as many times as preferable in a metaheuristic. The metaheuristic would, therefore, become very slow. It will be of limited use when not much time is available. This is the reason that no metaheuristic such as tabu search or simulated annealing is implemented. Instead, hill climbing is chosen, since for large instances it is already challenging to have a reasonable computation time for this simplest approach due to the edge disjointness constraint. The simple hill climbing is more likely to be able to find a better solution in limited time than a metaheuristic such as simulated annealing or tabu search. The latter is only worth implementing if there is a problem of getting in a local optimum really quickly, while still having much computation time left. To summarize: to meet the objectives of this research, it is not worth to consider something more difficult than hill climbing that is likely to even give worse solutions if the computation time is limited.

5.2 Hill climbing

In hill climbing an initial solution is improved by neighborhood moves until no further improvements can be made. In the previous section, the motivation for the choice for hill climbing was outlined. In this section, more details are given on the exact implementation. The initial solution can be suboptimal in three different aspects:

1. some street cabinets are not optimally clustered;

2. the order of the street cabinets in the rings is not optimal;

3. the disjoint routings between the street cabinets in a ring are not optimal.

These three are not always of equal importance. Which one is most important is very much influenced by the way the initial solution is constructed. It is easier to explain this statement by considering the insertion heuristic to show which aspects are most promising to improve upon for that case.

Recall that the decision how to initialize the rings for the insertion heuristic was already quite tough. Different options were studied, from which the "most spread" initialization performed better than the others for most instances, but still for quite some instances one of the other options performed the best. This clearly points out that for some instances some improvements are possible with respect to the first aspect. Another problematic decision is which of the street cabinets to insert first. Different rules with respect to the insertion order were tested to see which of them performed the best. From these options one was chosen that did not use too much computation time by avoiding computation of disjoint shortest paths at the risk of suboptimality. Summarizing, these are two decisions that were made based on incomplete information. Clearly, upon the decision which street cabinets to insert in which rings is still considerable opportunity for improvement of the initial solution. Therefore, the neighborhood moves based on relocation and exchange between rings are implemented.

With respect to the second and third aspect, only one concession to optimality is made during the insertion algorithm. When inserting a street cabinet in a ring all positions in that ring are considered and the cheapest option is chosen. Moreover, the computation which position is cheapest is done almost optimally. The only problem is that when inserting a street cabinet l in between SC i and j in a ring, the ring needs to be fixed and these edges cannot be used. This is unavoidable, since no algorithms are known yet to solve this problem optimally. Luckily, fixing the current ring does most of the time not seem to affect the quality of the routing. Using Suurballe's algorithm already avoids many inefficiencies in the routing. The deletion method described in Section 3.3 and Appendix A on the other hand, seems to hurt the quality of the routing a lot more. Summarizing, the use of Suurballe's algorithm combined with considering all possible insertion positions in a ring ensures already a very good solution quality with respect to the second and third aspect. Therefore, no neighborhood operations based on relocation and exchange within rings are programmed.

The neighborhood moves relocation and exchange of street cabinets between rings, can be implemented in various ways. Two options are investigated:

1. A consecutive implementation of relocation and exchanges.

2. An implementation that integrates the relocation and exchange in one.

The first option is particularly interesting when the instances are larger and not much time is available for the local search. Neighborhood searches for relocation moves are namely much faster than those for exchange moves. This has various causes. Firstly, there are less relocations than there are exchanges: $|SC| \cdot k$ possibilities versus $|SC| \cdot (|SC| - 1)$ options. Moreover, in the first case often the relocation is rejected immediately without computing disjoint paths, because the capacity of a ring would be exceeded if the SC is relocated. However, for many of the possible exchanges the capacity restriction would be fulfilled, since in both rings also an SC is deleted.

The second approach is likely to be able to find solutions of better quality than the first approach, provided that the alternation between relocations and exchanges is done in a sound manner. To decide which of the two types of moves is performed the following logic is used: candidate SC l_1 is chosen and for this SC the cheapest ring and position in all other rings with available capacity is calculated (disjoint insertion cost). Also the cheapest exchange with another SC l_2 from another ring is calculated for the rings where it was not possible to relocate SC l_1 to because of capacity restrictions. The rationale behind this idea is that it saves a lot of computation time, while at the same time the rings that are not considered for exchanges are still covered by the relocation moves.

Similar to the insertion heuristic, the decision remains in which order to consider the SCs. A random order can be chosen, but a more sensible option is ordering the SCs according to the the sum of the costs of the connections to their neighbors in the ring. The ordered list of street cabinets is run through and as soon as for an SC either a cost decreasing relocation move or a cost decreasing exchange move is found, the move is executed. An alternative would be to search the entire neighborhood and select the move that reduces the cost the most. This may lead to a better solution or less neighborhood searches. Searching the entire neighborhood each time would, however, take too much computation time. It is important to improve the solution as quickly as possible, since the maximal time reserved for improvements is, especially for larger instances, quite limited and the local optimum will often not be reached before this time bound is exceeded.

Finally, note that the local search can stop as a result of two situations:

1. No improvements can be found anymore. In other words, a local optimum is reached.

2. The computation time reaches its maximum (e.g. 5 minutes).

The second situation will occur for the larger instances, whereas for the small instances often a local optimum is reached earlier. A small worked-out example to illustrate the hill climbing is given in Section 6.2.

6 Test instances and a worked-out example

This section starts by describing the different types of instances that are used to test the insertion heuristic and local search methods developed in this thesis and the cluster first - route second method developed by TNO. Subsequently, in Section 6.2 a small instance is used to illustrate how the insertion heuristic (Section 4.2) and local search (Section 5.1) work.

In the testing no real world instances are studied, because these were not available within the time frame of this research. To be more precise, no data is currently available on existing ducts with available capacity. Information on locations of street cabinets and central offices and their characteristics are available for the Netherlands. It is possible to use real street pattern data and randomly decide in some way where ducts with available capacity are present. Due to the limited time available for this research, this testing is left open for further research. Similarly, testing on real world data from either the Netherlands or elsewhere is a logical next step to test the different methods and tailor them to perform as good as possible in as little time as possible. The focus in this thesis will be on theoretical instances that are based on a rectangular grid. These grids are, however, made in such a way that the essential characteristics of real world instances regarding the cost patterns are present. The grid defines a graph as follows: let the nodes be the points on the grid where two lines cross and let the edges be the lines connecting each node to its neighbor. The cost of each edge is chosen randomly (using an uniform distribution within some specified bounds). Next, part of the edges of the graph are reduced in cost (quite drastically) to represent the availability of ducts with available capacity. Different patterns of ducts are studied in this thesis, which are discussed in more detail in Sections 6.1.1 and 6.1.2. Next, part of the vertices of the graph are chosen randomly to have a special meaning. The most centrally located one in the grid will represent the central office, while the others will function as street cabinets. For each street cabinet the number of customers is chosen randomly according to some specified distribution. An example of a (small) grid instance is given in Figure 11. On the edges, the costs of the edge is indicated. Moreover, edges on which a duct is available with free capacity are drawn as solid lines, whereas when there is no duct available, but digging is allowed, a dotted line is used. The central office is marked by a circle and numbered one. The other street cabinets are numbered from 2 to 15.

6.1 Different types of grid instances

6.1.1 Grid instances with randomly chosen inexpensive paths

One of the duct patterns studied is where ducts lie in paths through the graph. In real world instances, it is this type of duct patterns that will often be present. Ducts may namely have been placed for other networks, such as the core network of mobile communications, the core network of the broadband internet or fiber cables going to business areas, etc. The instance given in Figure 11 is an example of such an instance where ducts are available on paths in the graph. To construct such paths, two points in the graph are randomly chosen (all vertices have equal probability) and the shortest path between them is used. In Figure 11, five of such paths were constructed. An example is the path from (1,8) to (2,5). Sometimes the paths are quite short and sometimes they are quite long, but because of randomness quite a mix of path lengths is chosen. Moreover, this allows a good study of the methods under various circumstances.

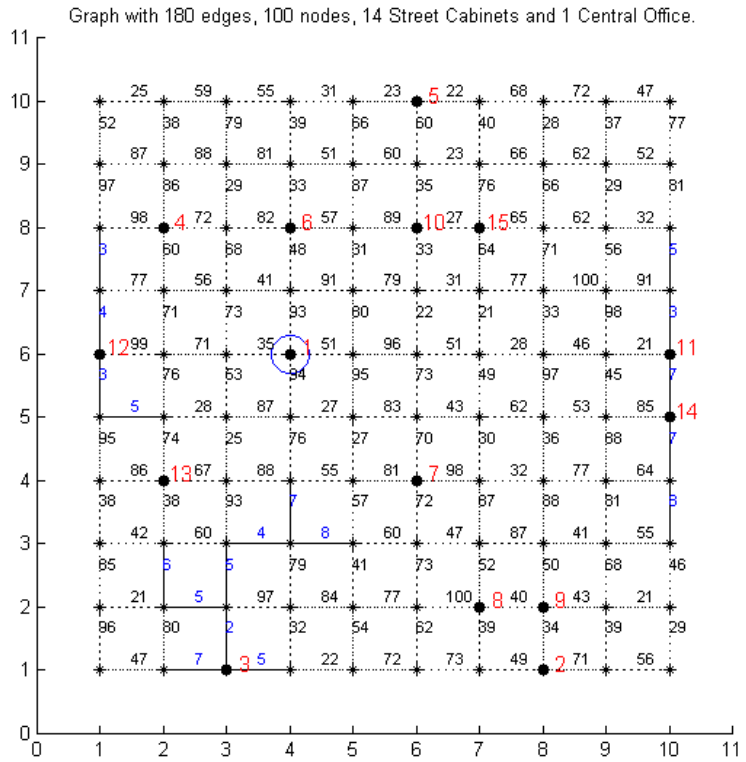


Figure 11: An example grid instance of size 10 by 10 with 14 street cabinets and a central office. Ducts are available on random paths in the graph.

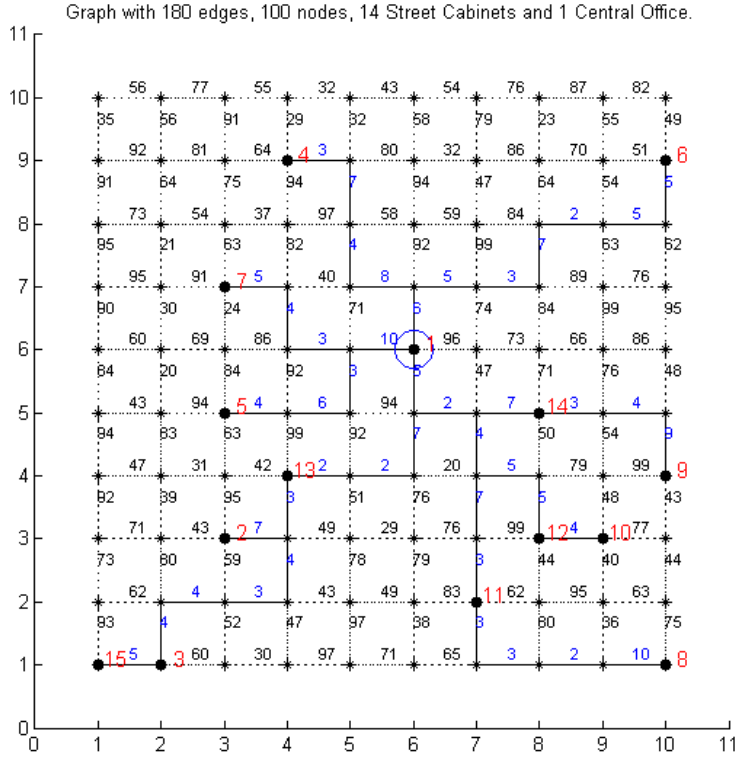


Figure 12: An example grid instance of size 10 by 10 with 14 street cabinets and a central office. Ducts are available on the shortest paths between each SC and the CO.

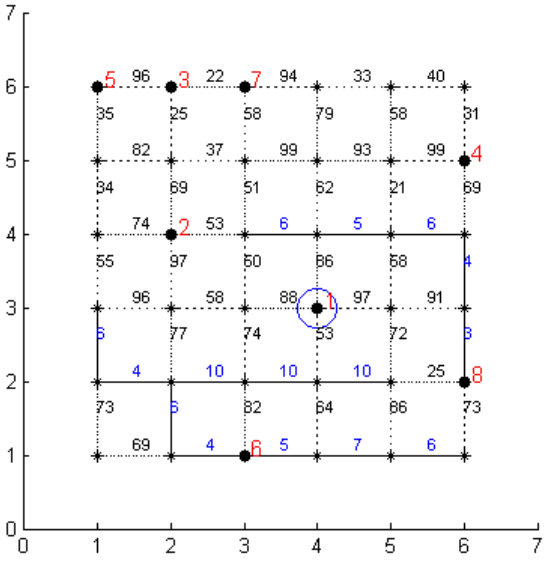
6.1.2 Grid instances with inexpensive paths from the CO to each SC

Another possible duct pattern is where ducts are available on a path from the central office to each street cabinet. In other words, a tree network of ducts is available. A major part of this network can be used to construct rings, so it is relatively inexpensive to create rings to ensure a high reliability. An example of such an instance is given in Figure 12. These paths were constructed by first determining the shortest path from the CO to each of the SCs using Dijkstra's algorithm. This type of duct pattern is closely related to the one studied by Kalsch et al. (2012). Their methodology is specifically targeted at embedding ring structures in large fiber networks.

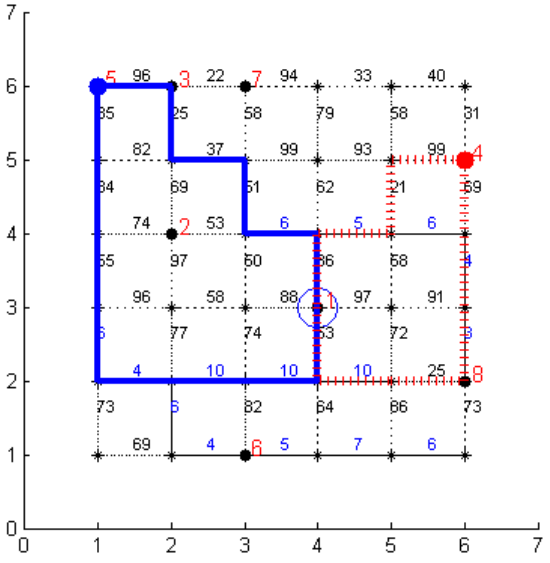
6.2 The insertion heuristic illustrated by an example

In this section the insertion heuristic is illustrated by a very small example to increase the reader's understanding of the algorithm. In Figure 13a an instance is given with 7 street cabinets and a central office. In this example, for illustrative purposes the number of customers of each SC is taken to one and the capacity of a ring equal to four. Two rings are needed to be able to insert all the SCs. The first ring is initialized by taking as a seed the SC furthest from the CO, which is SC 5 here. Then by Suurballe's algorithm two disjoint paths are constructed between the CO and SC 5. Next, the second ring is initialized by the SC furthest from the CO and SC 5. This is SC 4 and also for this SC two disjoint paths are calculated to the CO. The result is shown in Figure 13b. Ring 1 and ring 2 share common edges, but this is allowed; the edge disjointness constraint only has to hold within each ring. Then the non-disjoint insertion cost of the remaining SCs is calculated to decide on the next SC to insert. First, SC 6 is inserted, which can be done cheapest in ring 1 (solid blue line in Figure 13c). The next SC to insert is SC 8. This SC can be inserted for free, since it is already on the route of a ring. After that, it is the turn to SC 2 to be inserted (decided by non-disjoint insertion cost). The cheapest ring and position to insert this SC is calculated in a smart order (based on non-disjoint insertion cost). There are 6 possibilities, but due to the smart order Suurballe's algorithm only has to be used once! The first disjoint insertion cost is namely already smaller than all other non-disjoint insertion costs. Now, SC 3 is inserted; again with cost 0 (see Figure 13d). Finally, SC 7 is inserted. Ring 1 is already full. Of the remaining 3 insertion positions in ring 2 again only once disjoint shortest paths have to be computed. Now all SCs are inserted, and the final rings are shown in Figure 13e.

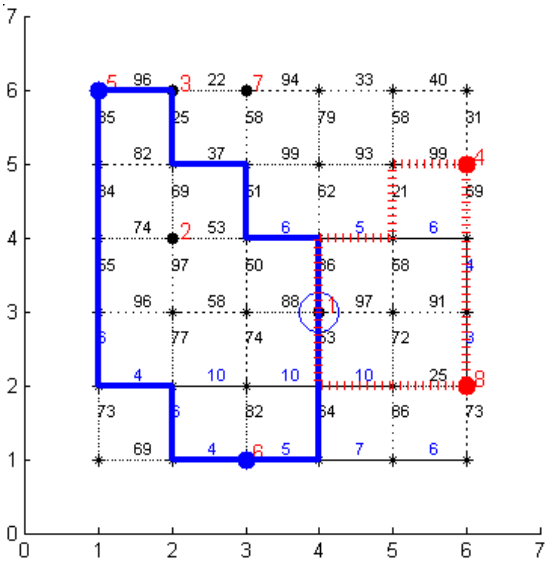
The total cost of the solution is 1161 euros. Note that the solution of the cluster first - route second method developed earlier by TNO (see Section 4.3) gives a solution that has 25% higher cost for this instance. Here this is caused mainly by a very inefficient routing as is clearly visible in Figure 14. Both solutions can, however, also be further improved using the local search described in Section 5.1. In particular, one cost reducing exchange move can be performed on the initial solution of the insertion heuristic. In Figure 15 this is illustrated. The hill climbing procedure starts by looking for relocations that save cost. Since ring 1 is used up to its capacity, only relocations from ring 1 to ring 2 are considered. However, it turns out that the cost of the solution cannot be reduced in this way. The next step is to consider the SCs in ring 1 to see if any exchanges with the SCs in ring 2 decrease the cost of the solution. The exchange between SCs 7 and 6 lowers the cost by 5.86% to 1093 euros. The results of this exchange is illustrated in Figure 15b. The solution of the cluster first - route second method can also be improved, but this solution gets stuck in a local optimum that is just a bit higher: 1140 euros.



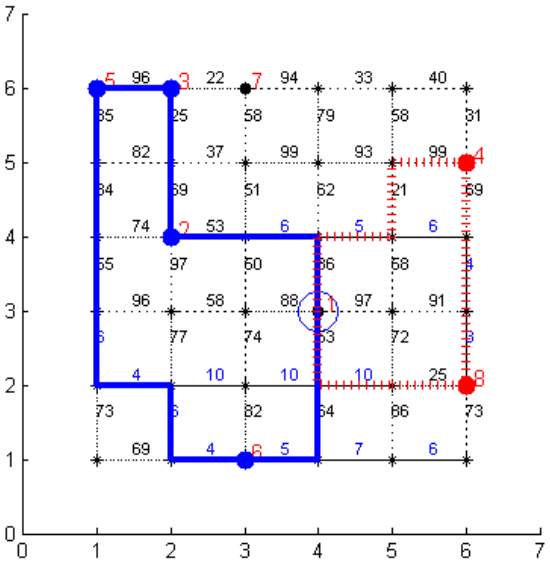
(a) Example instance.



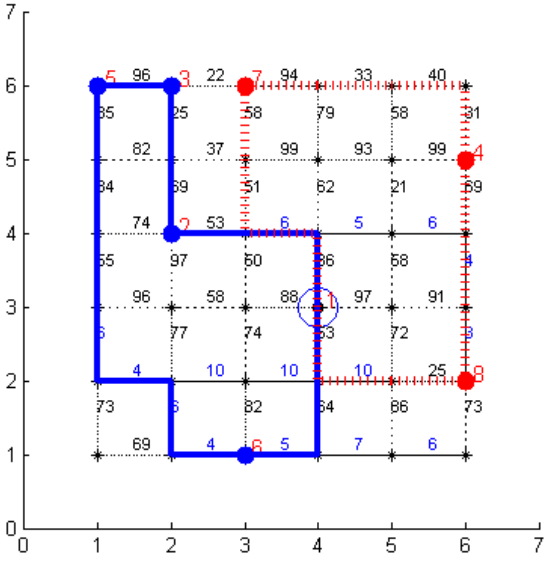
(b) Rings after initialization.



(c) Rings after insertion SC 6.



(d) Rings after insertion SCs 8, 2 and 3.



(e) The final rings.

Figure 13: A small example (6 by 6 grid with 6 SCs and a CO) to illustrate the insertion algorithm.

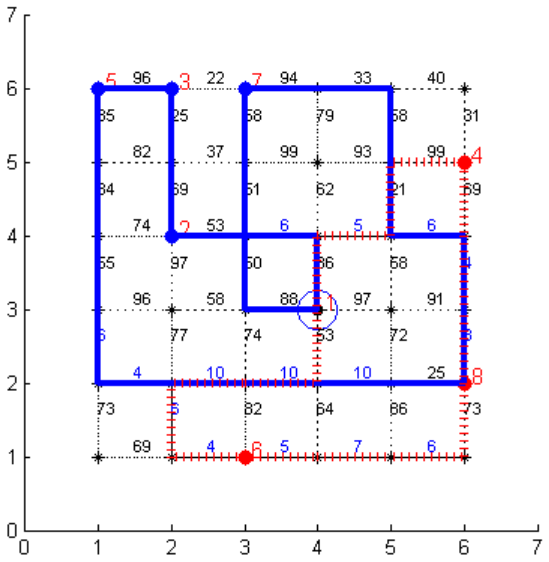
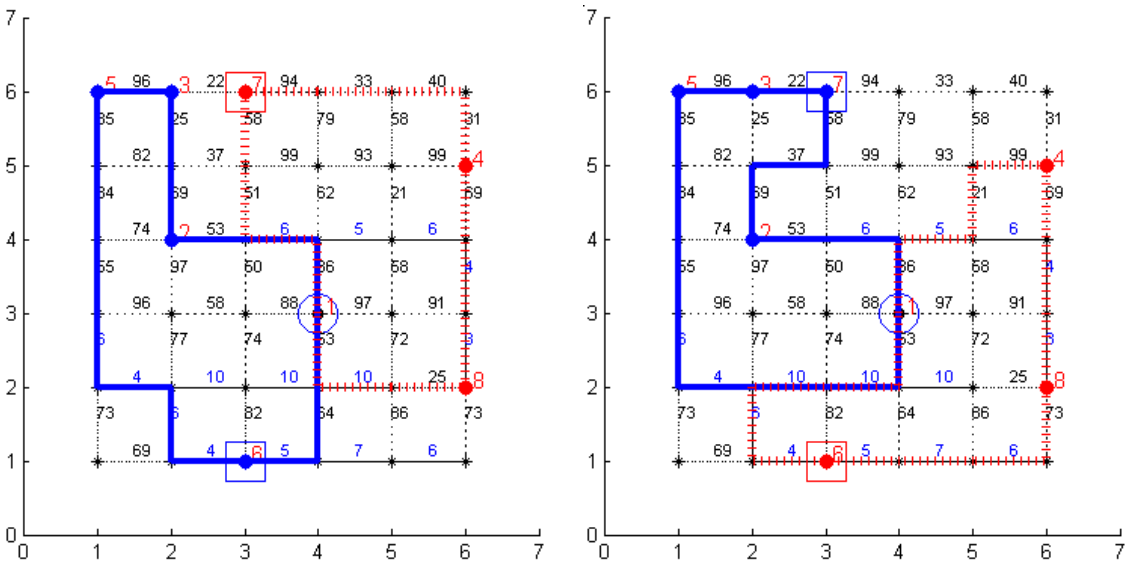


Figure 14: The solution of the cluster first - route second method developed earlier by TNO (see Section 4.3) for the example instance in Figure 13.



(a) Exchange SC 7 and 6.

(b) The rings after local search.

Figure 15: Hill climbing applied to the solution in Figure 13.

7 Test results

In this section different methods and tuning options are tested. None of these methods or tuning options that are tested completely outperform their alternatives for all generated random instances. Instead, three important performance criteria are defined to measure the performance of the different methods or tuning options over the different replications:

1. The proportion of times it finds the best solution (lowest cost) among the alternative methods or options.

2. The average cost over the different replications.

3. The average computation time over the different replications.

In the Tables 2-24 in this thesis these are written shortly in the headers as '% best', 'cost' and 'time', respectively. In some tables these performance measures are given at two distinctive time moments: after finding the initial solution and after local search. In the tables in Appendices B.1-B.4, each line corresponds to a different duct pattern. E.g. '20 cheap paths', indicates that in each replication 20 paths in the graph are chosen randomly to be inexpensive (start and endpoints are chosen randomly and the path in between is the shortest). 'CO to each SC' indicates that the shortest path from the CO to each of the SC is inexpensive. The amount by which the costs are reduced is mentioned in the header of the table.

This section continues by testing what are the best 'tuning options' of the insertion heuristic. In Section 7.1 four different seed selection methods are compared. Subsequently, in Section 7.2 three different types of insertion orders are tested. In Section 7.3 tests results are shown on the influence of the number of initial rings that is chosen. After having tuned the insertion heuristic, its performance is studied relative to the cluster first - route second heuristic developed by TNO. These results are discussed in Section 7.4. In Section 7.5 the local search procedure is tested. Finally, in Section 7.6 more attention is paid to the computation time.

7.1 Initialization of the rings in the insertion algorithm

In Section 4.2 different options were described to select seeds to initialize the rings. In particular, the first SC in each ring can be chosen randomly, based on the clustering algorithm of Phillipson (2013b) or by choosing SCs in such a way that they are most spread over the grid (according to cost). The second option has two variants: including the CO in the clustering or leave it out. These four initialization types are tested on instances differing in size, duct patterns, customer distributions, etc. To be able to draw a reliable conclusion random instances are drawn many times for each instance specification. Due to practical considerations, for larger instances less replications are performed, since the computation time of the testing would otherwise become too large to handle. Tables with the results are shown in Appendix B.1. Note that only the comparison within a line of a table is relevant. From the tables it is immediately clear that none of the four options outperforms the others for *all* random instances. For each instance (except one) of the type 'random inexpensive paths' the 'most spread' seed selection method clearly performs the best. Both the average cost and the proportion of all replications it finds the lowest cost consequently favor this type of initialization. An exception is the large grid (75

by 75) with no cheap ducts available at all. There 'centroid without CO', surprisingly, performs the best. In most instances from practice, cheap ducts are in fact available, but their amount differs per region. It is, however, unlikely that no ducts are available at all. Since the seed selection 'most spread' performs the best for all the from a practical perspective 'interesting' instance types, it is chosen henceforth as standard for instances of the type 'inexpensive random paths'.

For the instances where there is an inexpensive path from the CO to each of the SCs, the seed selection by using the 'Centroid without CO' has the best performance w.r.t. the average cost and '% best'. Only for the smallest instances, 'most spread' performs better than using the clustering algorithm of Phillipson (2013b) to get seeds for the rings. A little more computation time is needed to get the seeds, but this extra computation time is still reasonably small and it can be worth it to get a better solution quality. Therefore, for instances where all SCs can be reached via an inexpensive path, seed selection is, henceforth, done using the 'centroid without CO' method.

7.2 Insertion order

In the description of the insertion heuristic in Section 4.2 three different options were given concerning the insertion order of the SCs: randomly, based on non-disjoint insertion cost and based on disjoint insertion cost. These are the three basic options, which are tested on their performance. In the tables in Appendix B.2 the results are shown for various instance sizes and duct patterns. These show that the larger the instance (both in terms of grid sizes as in the number of SCs), the bigger the proportion of replications where the disjoint cost order leads to the lowest average cost ('% best'). For the instances of size 25 by 25 and 50 by 50, the percentage of times the disjoint cost order was the best over the different replications lies around 50% to 60%, whereas for the instances of size 75 by 75 is it closer to 90%.

The random order has the highest average cost for all tested instances. It is better to insert the SC with the lowest non-disjoint insertion cost each time. This does take a little more computation time, but leads to a lower average cost. Finally, the disjoint cost order has the lowest average cost, but the computation time is considerably larger. For the larger instances of size 75 by 75 it can take up to an hour to come to an initial solution, while the non-disjoint cost order is only using 2 to 3 minutes. The disjoint cost order does not satisfy the requirements set in the problem definition in Section 1.2: a computation time in the order of magnitude of a couple of minutes. Especially when observing that instances can even get larger (e.g. 100 by 100 with 600 SCs), it is clear that the disjoint cost order is no real option with the computation time restriction set in the problem definition. The reason that it is so slow compared to the other two options is that it involves computing disjoint shortest paths thousands of times. A single disjoint shortest path computation can be done in less than a second for the largest grids considered, but since it has to be done thousands of times, it becomes a problem.

Since the computation time of the disjoint cost order is already too large according to the requirements in this thesis, no effort is spent to improve upon the objective that decides which SC to insert first. Now it just chooses the cheapest SC, but probably the solutions can be further improved in two ways: firstly, by using information on how expensive it is to insert the SCs in alternative rings, it can be avoided to some extent that some SCs are left in the end

to insert in rings very far away, while the rings nearby are already full. Another way to avoid these situations is to artificially increase the insertion cost when a ring becomes fuller. The former option might be hard to incorporate in the non-disjoint cost order, since these costs are not that precise and can be deceiving with respect to the actual disjoint insertion cost. The second option is much more promising for the non-disjoint cost order. Both options can also be implemented in the disjoint cost insertion order, which are both expected to lead to lower cost in that case. However, the computation time will remain (too) large. The implementation and testing of these two options is left open for further research. The non-disjoint cost order is chosen as standard in the rest of this thesis.

7.3 Influence of the initial number of rings

An interesting question raised in Section 4.2 is: can the cost of the insertion heuristic be lowered (on average) if a certain percentage of extra rings is added from the start to the lower bound $k = \lceil \sum_{i \in SC} q_i / Cap \rceil$? To be able to answer this question, in Appendix B.3 the results of the insertion heuristic for large instances are shown for a different number of initial rings. From the different tables it can be observed that the average costs of the different number of rings are pretty close to each other. Still, it is worth starting with a high number of initial rings as is visible from the tables. Instances with different characteristics are used to be able to get a good overview. It takes, however, a lot of time to compute initial solutions for so many different initial rings. This makes it hard to come to a conclusion that is very reliable. An attempt is made, but much more testing would have been preferable; both concerning instance types as the number of replications. The results give, however, some indication.

There are different possibilities to decide for an instance what is the best number of initial rings. The following option is chosen: for each replication compute the optimal percentage of extra rings (based on the initial solution). Next, take the average over all replications. This average percentage is given for each instance at the bottom of the table. For instance, for the instance in Table 11 this is 15.49%, which boils down to approximately two extra rings. To decide about the number of extra rings in general, the average is taken over these percentages of all test instances. This gives a percentage of around 14%, which will be used in the rest of this thesis. Another option is to choose the initial number of rings based on the lowest average cost over all replications and estimate the corresponding percentage of extra rings. This may lead to a slightly different number of rings, but often the difference between the two methods is very small.

Yet another possibility is to incorporate also the cost after improvement into the decision. One could imagine that depending on the initial number of rings it might be harder or easier to improve the solution. For example, a small number of initial rings may have the disadvantage that it is harder to improve the solution. Since the rings are very full in that case, relocations are possible less often. On the other hand if too much rings are chosen, it might be hard to get rid of all of them in the local search, since a ring will only be become redundant if it is completely empty, a thing that may not always happen. Since the available computation time is limited (e.g. 5 minutes), for the larger instances often no local optimum is reached yet when the local search is aborted. This really pleads choosing an extra number of rings in such a way that the initial solution cost is as low as possible. The results in Tables 9 to 14 show that this is the case for the mentioned 14% extra rings.

At the same time it is expected that when no limits would be put on the computation time, it might be wiser to choose no or a very limited number of extra rings, since it is probably hard to get rid of rings. It is, however, almost impossible to test this in a good way, since for many different number of extra rings local search would need to be run much longer (hours instead of minutes). Automatically, it is then no longer possible to do multiple replications, making it hard to see if it is just coincidence that a certain number of rings is the best or not. Two replications are run of one problem instance (the same as in Table 13) to get a general idea whether this hypothesis might be true (see Table 15 in Appendix B.3). The time available for the local search is limited to 100 minutes, which is significantly more than than the 5 minutes in the other tests. The results of the two replications show that the improved cost is lowest for 0 or 1 extra rings. This is clearly different from the results with short improvement time, where the best number of extra rings of the lowest improved cost was often the same as the best number of extra rings of the lowest initial cost.

Summarizing, when little time is available for the local search, the best results are obtained if a certain percentage of extra number of rings is chosen. Based on the testing in this thesis, the optimal percentage of extra rings is estimated to be around 14%. However, when much computation time is available, preliminary tests show that it might be wiser to choose no (or very little) extra rings at the start.

7.4 Comparison with cluster first - route second heuristic

The most interesting part of the testing of the insertion heuristic (Section 4.2) is the comparison with the cluster first - route second heuristic (Section 4.1.3). These test results can be used to answer the main research question in this thesis as was formulated in Section 1.2: does integrating the clustering and disjoint routing into one approach give better solutions than a separated approach? Before turning to the test results, note that it is almost impossible to draw a general conclusion on this question. Many options and tuning parameters are present in each of the approaches. What can be compared are the results of the cluster first - route second approach as implemented by TNO and the insertion heuristic developed in this thesis. In both algorithms modeling choices have been made that possibly could be improved upon. This is exactly the reason why in the recommendations for further research a suggestion is made on how to integrate the 'best' of each method into one method, which might lead to better results than each of them separately.

The tables with test results are given in Appendix B.4. In Table 1 an overview is presented that contains a comparison in terms of percentages of the average cost of the initial solution between the insertion heuristic and the cluster first - route second heuristic. Among all tested instances, the insertion heuristic gives an average cost that is between 27% lower and 3% higher than the average cost of the cluster first - route second heuristic of TNO.

7.4.1 Initial solution

First the initial solutions of both heuristics are compared. The analysis starts with the results of the instances where ducts are available as random paths through the graph (see Section 6.1.1). Afterwards, the test results for the instances where ducts are available from the CO to each of

Table 1: Overview of the test results: the percentage that the insertion heuristic is better (in terms of average cost) than the cluster first - route second heuristic by TNO.

Instance characteristics						**% insertion heuristic**
grid	**# SCs**	**cap**	**cust**	**cost red.**	**# cheap paths**	**cheaper than CF-RS**
10x10	15	2500	DU(100,200,300,400,500)	90%	10	12%
					5	8%
					0	1%
					CO to each SC	26%
20x20	50	8	1	50%	20	2%
					10	1%
					0	-2%
					CO to each SC	3%
30x30	100	2500	DU(100,200,250,300,400)	80%	30	9%
					15	6%
					0	-1%
					CO to each SC	12%
50x50	100	2500	DU(100,200,300,400,500)	60%	50	1%
					25	2%
					0	-3%
					CO to each SC	1%
60x60	50	2500	DU(100,200,300,400,500)	85%	45	2%
					20	1%
					10	-1%
70x70	250	2500	DU(100,200,300,400,500)	75%	60	12%
					30	13%
					10	10%
					CO to each SC	16%
85x85	350	12	1	80%	75	2%
					40	2%
					CO to each SC	10%
95x95	500	2500	DU(100,200,300,400,500)	85%	20	23%
					10	19%
					0	9%
100x100	600	2500	DU(100,200,300,400,500)	90%	100	21%
					50	25%
					25	27%

the SCs, are treated.

The performance of the insertion heuristic is quite good compared to the cluster first - route second heuristic. When looking at the percentages in Table 1, it can be concluded that the insertion heuristic always finds a pretty good solution compared to the cluster first - route second heuristic. Only for four instances the performance of the cluster first - route second heuristic is better (in terms of average cost) than that of the insertion heuristic. The difference is, however, not that large: only 1% to 3%. Three out of these four instances are medium-sized instances where no cheap ducts are available at all. In most practical instances these will, however, be present. Even when building new cities or districts, ducts will often be available, since other networks have to be installed too. The costs of the ducts can then be shared between different network operators. The fourth instance where the cluster first - route second heuristic performs a little better is the instance of size 60 by 60 with relatively very few SCs and only ten inexpensive paths. In the solution after the local search the roles are reversed though; there the solution originating from the insertion heuristic is just a little bit better on average. Whereas for medium-sized instances with very few or no inexpensive paths, the two heuristics perform quite similarly, for large size instances with no or few inexpensive paths the performance of the insertion heuristic is significantly better than that of the cluster first - route second heuristic. In fact, for the instance of size 95 by 95 with 500 SCs (Table 23) for all replications the solution of the insertion heuristic was the lowest. In terms of average cost over the different replications the insertion heuristic performed between 9% and 23% better.

For instances where cheap ducts are available, the insertion heuristic is better able to use this to its advantage. In fact, the insertion heuristic was developed with this goal in mind: integrating the decision of the clustering and routing. Whereas the cluster first - route second heuristic does not take the cost information into account when clustering, the insertion heuristic does. Combined with a routing that is closer to optimal, this explains the significant cost reductions that the insertion heuristic can give compared to the cluster first - route second heuristic.

The instances studied can be divided in two categories with respect to the customer information used. The first category is the special case where each SC has equal weight. In this case the restriction on the capacity of a ring reduces to having a maximum number of SCs in a ring. In the general case, SCs can have multiple customer weights. The results show that for the special case the performance of the two heuristics is closer to each other, though still in favor of the insertion heuristic. The computation time is also bit lower for the insertion heuristic for the instances with random inexpensive paths. On the other hand, for the general case, which is closer to reality, the insertion heuristic shows much better results, especially for large instances. For example, for the instances of size 70 by 70 and of size 100 by 100, the insertion heuristic gives solutions with an average cost between 10% and 27% lower. Also the proportion of the times the insertion heuristic is the best is 100% here. For smaller instances, this proportion is more around 60% for the insertion heuristic versus 40% for the cluster first - route second heuristic. The computation time of the insertion heuristic is, however, 24% larger on average. Surprisingly enough, the insertion heuristic is faster than the cluster first - route second heuristic for the instances where each SC has weight one; also for the large one of size 85 by 85.

The instances where an inexpensive path is available from the CO to each of the SCs give very clear results: the insertion heuristic performs the best, both in terms of average cost as in the proportion of the replications it gives the best solution. The percentage the insertion heuristic has a lower cost with respect to the cluster first - route second approach is rather high. The

only exception is the instance of size 50 by 50. This can probably be explained by a relatively small percentage of cost reduction (only 60%) combined with a low ratio $\frac{|SC|}{|V|}$. These two things reduce the impact of the disadvantages of the cluster first - route second approach.

Another advantage of the insertion heuristic is that it always finds a solution if there exists one. The cluster first - route second heuristic developed by TNO does not have this property. The test results showed that the larger the ratio $\frac{|SC|}{|V|}$, the more often no solution is found. The ratio varied for the studied instances between roughly 1% (instances of size 60 by 60) and 15% (instances of size 10 by 10). For most tested instances, this ratio is around 5%, which also seems more or less the boundary where the cluster first - route second heuristic switches from only very occasionally (in $0\% - 2\%$ of the cases) finding no solution to quite often ($2\% - 5\%$ of the cases) finding no solution. The insertion heuristic always finds a solution, making the heuristic more generally applicable. Thinking of a city with street cabinets, ratios of 5% or above do not seem that uncommon, since street cabinets are often present in many streets. Just to give the reader an indication: when looking at the centre of Amsterdam over 500 SCs are present, while in the street data from the land registry (Kadaster, 2013) the number of vertices for this area is roughly between $5,000 - 10,000$, leading to a ratio of around 5% to 10%. This estimate is a rough one, because in the data still vertices are present which are not strictly necessary in the computation. It still contains vertices that are present to indicate bends in the road, instead of junctions between roads. As mentioned in Section 6, the testing with these kind of data is left open for further research because of the limited time available for this thesis.

7.4.2 Solution after local search

After analysing the results of the initial solutions, now the effects of applying the local search are investigated. Note that the computation time of local search was restricted to 5 minutes, since the practical considerations mentioned in the problem definition in Section 1.2 require the total computation time to be in the order of magnitude of minutes. Only for the smallest two instances the local search stopped in less than 5 minutes because a local minimum was reached. In all other instances, it was the maximal time which lead to the end of the local search. It might surprise the reader that the average computation time of the local search exceeds the threshold 300 seconds in the tables in Appendix B. The local search was programmed in such a way that once the maximal time is exceeded, the computations on possible relocations and exchanges of the current SC are allowed to be finished. For the larger instances this can take up to a little over a minute.

Since the local search is often aborted before the local minimum is reached, the initial solution is very important. The results show that a lower cost of the initial solution leads to a lower cost of the local search. This is partly caused by the abortion, but the smaller instances where a local optimum is reached show that this cannot be the only reason. There the insertion heuristic reaches a lower local optimum than the cluster first - route second approach. Moreover, it also needs less time to do so. From this it can be concluded that the insertion heuristic is less easily stuck in a local optimum. The reason is that for the insertion heuristic often a few SCs are wrongly placed, whereas in the cluster first - route second approach the solution may need to be changed more drastically if it turns out that the clustering is done in a bad way. Moreover, some inefficiencies in the routing may persist in the solution that originated from the initial solution of the cluster first - route second heuristic.

7.4.3 Performance w.r.t. the optimal solution

No exact approaches were implemented in this thesis. This has several reasons: firstly because of limited time available for this thesis and secondly because it will only be able to solve very small instances. Possibly, even the smaller instances from the test results would already take longer to solve exactly using an ILP solver than desired. The computation times of exact approaches for related problems studied in other papers (see Section 2) are, namely, already too large for smaller instances than are considered here. As a consequence of not having implemented an ILP approach, no conclusions can be drawn on the performance of the heuristics compared to the optimal solutions. Also when looking at the results it is not always easy to judge on the quality of the solution, since the cost and duct structures that are used prevent that. The newly developed insertion heuristic can only be compared to the cluster first - route second heuristic of TNO. The testing shows that in terms of the quality of the solutions the insertion heuristic is clearly preferable. The computation time is somewhat larger, but is still within the set limits.

7.5 Performance of local search

In Section 7.4 already some remarks were made about the performance of the local search. Due to the computation time restriction from the problem definition, the local search was performed only for 5 minutes. In this section the local search is run longer for a few instances to see how the solution improves over time. In Section 5 two options were mentioned: an integrated approach and a consecutive approach. Which of the two performs the best, is not an easy question. A lot of testing is needed to answer this question in a sound manner. This is left for further research. Here only some preliminary tests results are presented for three instances. Some of the results confirm the conjecture that the integrated approach is able to find a better quality solution, but initially the consecutive approach improves the solution faster. For one of the tested instances, however, the consecutive approach led to a lower final cost. From this it is at least clear that it is not so easy to draw a general conclusion without extensive testing. Note that the testing in Section 7.4 uses the integrated approach. Those test were performed before the preliminary test in this section. This is, however, not that much of a problem, since the main focus in that section was on the initial solution anyway.

Which of the two approaches performs best probably also depends on the characteristics of the instance. For example, if all rings are quite full, one might imagine that exchanges might be more important in an early stage to make any improvement at all. In that respect the number of extra rings taken in the initial solution plays an important role. As was noticed in Section 7.3, the time allowed for the improvement is also a very important factor in the decision how many extra rings should ideally be taken. The more time spent on the improvement, the fewer extra rings should be taken. In that case it is expected that the integrated approach works better, since exchanges are considered earlier.

In Figures 16-21 the initial solution is improved by both local search approaches. Each time the progress of the improvements over time is studied starting from both the initial solution of the insertion heuristic as from the initial solution of the cluster first - route second heuristic. The maximal time of improvement is taken to be 2 hours, instead of 5 minutes. The parameters are the same as for the corresponding instances in the tables of Appendix B.

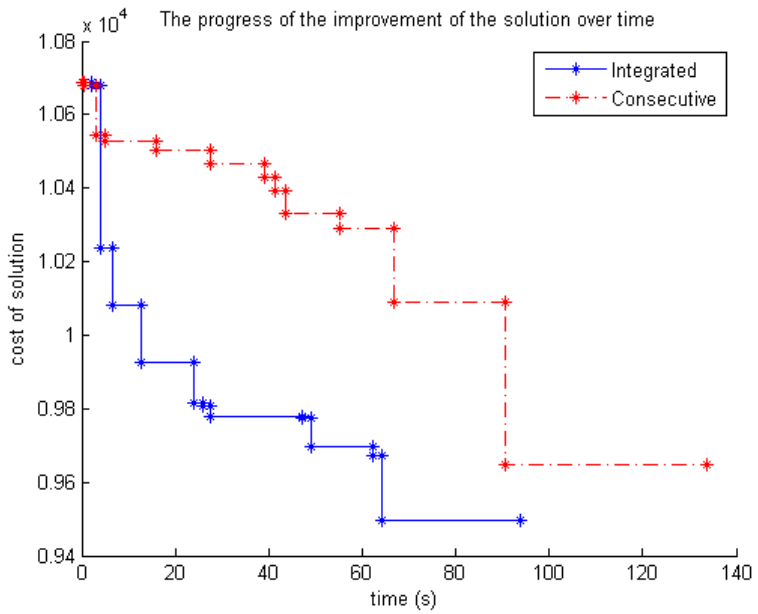


Figure 16: Comparison of local search approaches on the initial solution of the insertion heuristic for an instance of size 20 by 20 with 50 SCs.

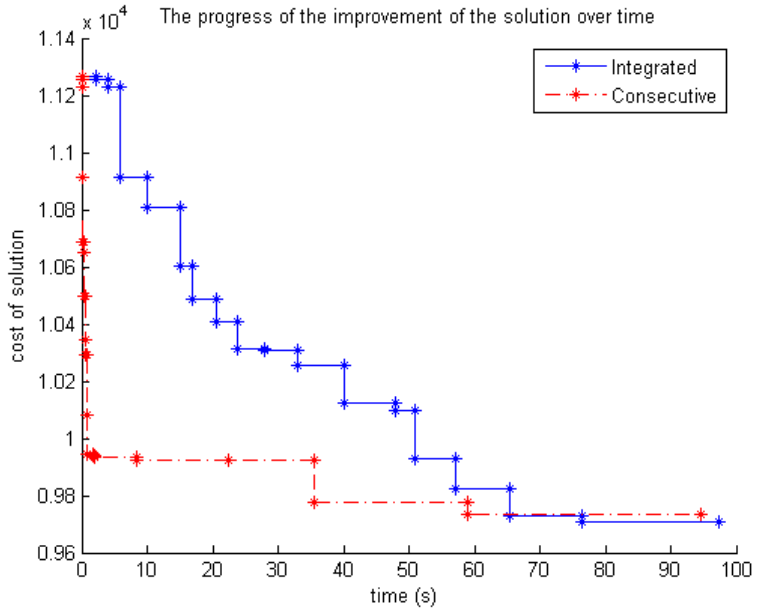


Figure 17: Comparison of local search approaches on the initial solution of the cluster first - route second heuristic for an instance of size 20 by 20 with 50 SCs.

In Figure 16 the initial solution of the insertion heuristic is improved for an instance of size 20 by 20 with 50 SCs. The number of cheap paths is equal to 10. The local search stops because a local optimum is reached. In this case the integrated approach finds better solutions more quickly than the consecutive approach. The integrated approach improves the initial solution by 11.14%, whereas the consecutive approach improves the initial solution only 9.72%. For the same instance also local search is run starting from the initial solution of the cluster first -

route second heuristic (see Figure 17). Here the consecutive approach reduces the solution more quickly initially, but in the end reaches a little worse solution than the integrated approach. In the end, the initial solution is improved with 13.82% by the integrated approach and with 13.58% by the consecutive approach.

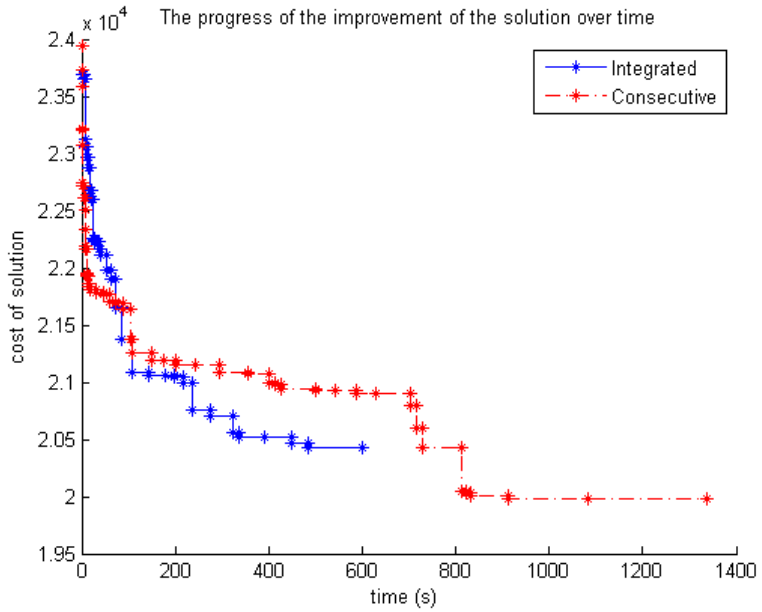


Figure 18: Comparison of local search approaches on the initial solution of the insertion heuristic for an instance of size 30 by 30 with 50 SCs.

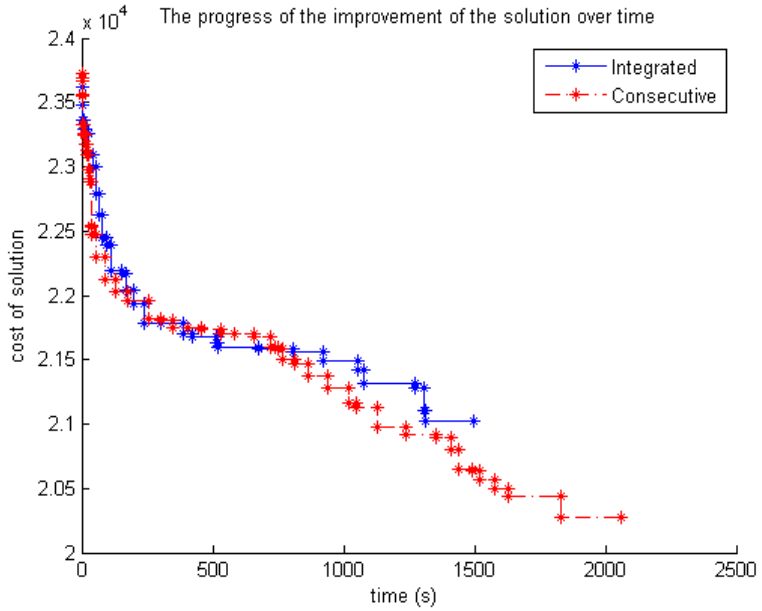


Figure 19: Comparison of local search approaches on the initial solution of the cluster first - route second heuristic for an instance of size 30 by 30 with 50 SCs.

For an instance of size 30 by 30 with 50 SCs (15 cheap paths) the roles are reversed. The initial solution of the insertion heuristic is improved with 14.69% by the integrated approach, whereas the consecutive approach improves it by 16.57% (see Figure 18). When starting with the initial

solution of cluster first - route second heuristic (see Figure 19), the consecutive approach also reaches a lower local minimum. The integrated approach realizes an improvement of 11.41% and the consecutive approach of 14.56%. The conjecture that the integrated approach reaches a better local minimum is in this case not confirmed.

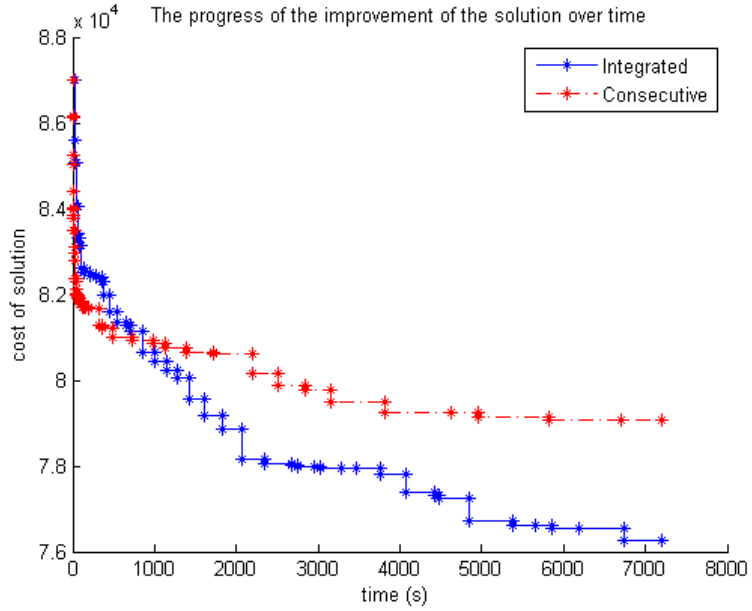


Figure 20: Comparison of local search approaches on the initial solution of the insertion heuristic for an instance of size 70 by 70 with 250 SCs.

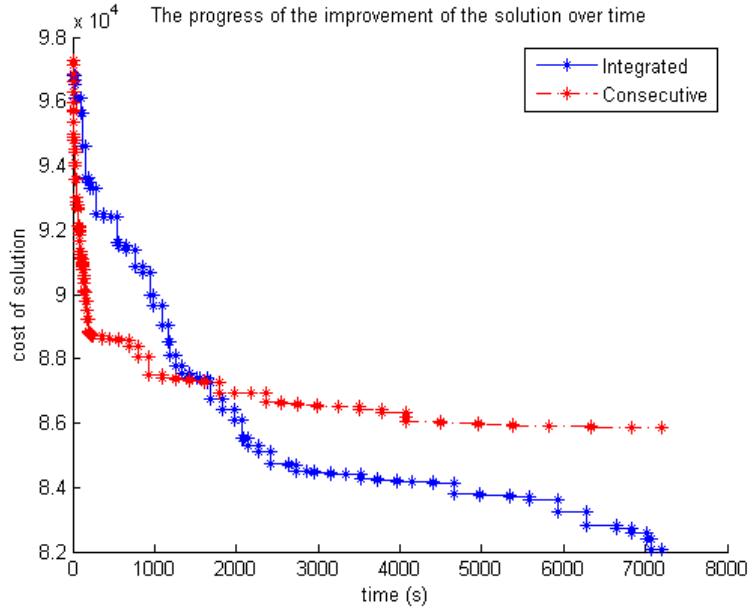


Figure 21: Comparison of local search approaches on the initial solution of the cluster first - route second heuristic for an instance of size 70 by 70 with 250 SCs.

Finally, a larger instance of size 70 by 70 with 250 SCs is studied. The number of cheap paths is chosen to be 30. Starting from both the initial solution of the insertion heuristic as from the initial solution of the cluster first - route second heuristic, the conjecture is confirmed

that the integrated approach finds solutions with lower cost, but that initially the consecutive approach improves faster since relocations are faster to compute. The progress of the local search is visible in Figures 20 and 21. In all cases the improvement ended because the maximum time was exceeeded and not because a local optimum was reached. The initial solution of the insertion heuristic is improved by 12.34% and 9.15% for the integrated and consecutive approach, respectively. For the initial solution of the cluster first - route second approach the improvements are 15.63% and 11.76%, respectively.

7.6 Computation time

In the problem definition it is argued that the preferred computation time is in the order of magnitude of minutes. One of the reasons was that the method should be implementable in a tool that allows for interaction with the user. The test results show that the computation time to find an initial solution satisfies the restrictions. For the largest instances, the computation time remained within approximately 12 minutes on the somewhat slower computer (see Table 24). On the faster computer that was used for an instance that is just a little smaller, the computation time was even around 5 minutes (see Table 23). These two instances contain the characteristics of the largest instances that need to be solved in the Netherlands: an area of a little over 500 SCs that need to be connected to one CO in an underlying graph with around $20,000$ edges.

Using the 'profile' option in Matlab, it can be discovered what lines in the code are taking the longest to evaluate. In this way the bottlenecks in the code can be identified and possibly resolved. For the largest instance the profiler indicated the following: Dijkstra's algorithm is called over $25,000$ times. Some lines within Dijkstra's algorithm are passed over a billion times! Approximately 91% of the total computation time of the insertion heuristic is spent on Dijkstra's algorithm, which clearly makes it the bottleneck. The implementation of Dijkstra's algorithm that is used (by Gleich (2009)) is, however, already very fast compared to its alternatives available online, since it uses binary heap. The only remedy to speed up the heuristic is, therefore, to use Dijkstra's algorithm less often. Several order tricks were introduced in this respect (see Section 4.2), which were very successful.

The smaller instances with up to 50 or 100 SCs can be solved within a few seconds. The computation time grows as soon as the number of SCs increases and as the size of the grid increases. The effect of the latter is a bit dampened by deleting an irrelevant part of the graph when computing shortest paths. This has to be done in a conservative manner, since otherwise the best possible solution may not be found. The number of SCs has a much larger impact, since not only more SCs need to be inserted, but also the number of possible insertion positions increases rapidly, since the number of rings is larger. In that sense, 500 to 600 SCs is about the maximum number of SCs that can be considered while staying within roughly 10 minutes of computation time. Luckily, this is also the maximum number of SCs that is observed in practice.

When comparing the computation time of the insertion heuristic with the computation time of the cluster first - route second heuristic an interesting property is observed. The computation time of the insertion heuristic is quite dependent on the duct pattern that is used. The fewer cheap ducts are available, the faster the computation of the initial solution becomes. This is

particularly easy recognizable for the instance of size 95 by 95 (see Table 23): having no inexpensive paths versus 20 inexpensive paths differs a factor 2.5 in computation time. The cluster first - route second heuristic is, however, almost insensitive in this respect; the computation time is hardly affected by the duct pattern. Whether this is an advantage or disadvantage depends on the context, but at least for this instance with 500 SCs with no inexpensive paths, the computation time of the insertion heuristic is significantly smaller than that of the cluster first - route second heuristic, while at the same time obtaining a 9% better solution. Difference in computation time between the two heuristics varies quite much; the insertion heuristic is between -67% and 63% slower than the cluster first - route second heuristic for the tested instances. The cluster first - route second heuristic is, however, in most of the cases faster: on average 24%. An exception is for the special case where each SC has equal customer weight. Then the insertion heuristic is up to 30% faster.

The computation time of the local search is much more problematic. Only for the smaller instances a local optimum is reached within 5 minutes. For larger instances this can take up to hours and for the largest instances maybe even a few days. Some tricks might still be able to reduce the computation time of the local search a little bit. Perhaps by developing techniques that improve upon the order of the SCs that is considered for exchange or relocation. Another possibility is maintaining a list with recently unsuccessful relocations and exchanges, which could also save some computation time. This can be investigated further in future research.

8 Conclusions and recommendations

8.1 Conclusion

In this thesis the design of a telecommunication network was studied, in particular the roll-out of Fiber to the Cabinet. This network has to satisfy several restrictions. The network should be ring-shaped, where each ring (circuit) should be edge disjoint and has a fixed capacity. A network is available that contains two types of edges: edges with available ducts (relatively inexpensive) and edges where excavations are permitted (relatively expensive). The problem is NP-hard, since it is a generalization of the Capacitated Vehicle Routing Problem (CVRP), which is shown to be NP-hard (Lenstra and Rinnooy Kan, 1981). Since the targeted computation time was in the order of magnitude of minutes, no exact models were implemented, but instead heuristics were considered. The same problem was investigated before by TNO (i.a. Phillipson (2013b) and Phillipson (2013c)). They used a cluster first - route second heuristic to find a solution to the problem. Their approach has the advantage that it is easy to understand and relatively fast. At the same time their method has some serious disadvantages. Firstly, their clustering is based solely on Euclidean distances and not on the real costs in the network; the information on the inexpensive ducts is not used. Secondly, in the routing the 'deletion method' is used, which regularly leads to serious inefficiencies in the routing and furthermore sometimes causes the whole method to find no solution at all.

The main research question of the thesis was the following: does integrating the clustering and disjoint routing into one approach give better solutions than a separated approach? In this respect an insertion heuristic was developed to find an initial solution. Subsequently, this solution is improved using local search, until either the maximum time is exceeded or a local optimum is found. The insertion heuristic starts by selecting seeds for all rings and then greedily inserts all street cabinets in the different rings. Different seed selection methods and insertion orders were implemented and tested. The best way to choose the seeds turned out to be iteratively selecting SCs that are as far as possible from each other (in terms of the cost in the graph). Computing all disjoint insertion costs for each unconnected SC each time, leads to the best results, but is too slow to remain within a computation time of at most several minutes for large instances of up to $10,000$ vertices, $20,000$ edges and 600 SCs. Therefore, non-disjoint insertion costs are used to determine the insertion order. Furthermore, the influence of the initial number of rings was investigated. The analysis showed that if little time is available for the local search, it is better to choose extra rings for the initial solution. If much time is available for the local search, however, it is wiser to choose no or only few extra rings.

After tuning the different options in the insertion heuristic, it was tested against the cluster first - route second heuristic by TNO. The insertion heuristic showed very good performance compared to the cluster first - route second heuristic. Only for a few tested instances, the cluster first - route second heuristic showed lower average cost (1% to 3%) over 100 replications: medium-sized instances with no or very few inexpensive ducts. For small and large instances with no or few inexpensive ducts and instances with more inexpensive ducts, the insertion heuristic gave solutions that were on average between 1% and 27% better. The computation time of the insertion heuristic is in most cases larger, but remains well within the set limits. A big advantage of the insertion heuristic is that it always finds a solution, whereas the cluster first - route second heuristic regularly fails to give a solution especially when the number of SCs is a bit higher compared to the number of vertices in the graph.

The local search can improve the initial solution of both heuristics, but since it takes very long to reach a local optimum the quality of the initial solution is extremely important. To answer the main research question: an approach that integrates the clustering and routing decision gives better results, while the computation time is a bit larger in most cases, but still within the set limits.

To finish, this thesis' contribution to the literature is highlighted. It is innovative in the use of Suurballe's algorithm in network design problems with edge disjointness restrictions. To the best of knowledge of the author, all previous work focussed either on exact approaches that require very large computation times for large instances or on heuristics that can seriously affect the routing quality by iteratively constructing a route (Kalsch et al., 2012) or using the 'deletion method' (TNO). Moreover, this thesis is a useful addition to the literature on VRP heuristics, since it is the first to integrate the complicated edge disjointness constraint successfully into the known methods. Finally, it clearly outperforms the currently used cluster first - route second heuristic for the roll-out of Fiber to the Cabinet.

8.2 Recommendations for further research

The research in this thesis answered some important research questions and gave insight in the problem. There are, however, still many possibilities and angles that can be investigated further. Below these are described shortly. Firstly, the methods can be tested on grid instances in which the vertices are not standardly connected to each of its neighbors. In reality, some vertices have more connections than others. Moreover, obstacles (e.g. canals or creeks) can cause having only few possibilities to go in a particular direction. This could have a significant effect on the performance of the methods. For example, the cluster first - route second method may run in serious problems when constructing routes using the deletion method. It is expected that more often than in the instances studied in this thesis, no solution or solutions with very high cost are found by this method. The insertion heuristic is very likely to be less sensitive to such problems, since the routing is done in a reliable fashion that always finds a good solution. After testing on such instances, by testing on real-world graphs (with actual duct information) a final conclusion on the performance of the different methods can be reached.

Besides more extensive testing, the different methods themselves also provide opportunities for further study. For applications where computation time is less important, it is expected that the initial solution quality can still be improved by using disjoint insertion cost information for all SCs for each ring to decide on the insertion order. The results from Section 7.2 show great potential in this respect. Sophisticated objectives that decide which SC to insert first, could be used instead of just inserting the cheapest SC. In this way, relatively expensive insertions at the end of the algorithm can be prevented a lot better. Note, however, that this is only possible at the expense of significantly larger computation time.

Another option is to increase the insertion cost artificially as the rings become fuller. In this way, as a ring becomes fuller only SCs that have no relatively inexpensive insertion alternatives will be inserted. This could prevent that the last insertions are very expensive because the nearest rings are already fully used. To get the right parameter values for the artificial cost increase, extensive testing will be required though.

As already mentioned in Section 7.6, the computation time of the local search is quite long. Future research can try to speed it up by looking further into the order of the SCs that is considered for exchange or relocation. Additionally, maintaining a list of recently unsuccessful relocations and exchanges to temporarily skip them, can probably increase the convergence speed. Furthermore, it can be more extensively tested which of the two approaches, an integrated or consecutive approach, is preferred in general. In this thesis only some preliminary tests were performed in Section 7.5.

Finally, it is interesting to see the result of combining different aspects of the cluster first - route second heuristic and the insertion heuristic. The following method is suggested: cluster the SCs using Phillipson (2013b) (including the CO), where the clustering should, however, not be based on the Euclidean distances, but on real cost in the graph (including the inexpensive ducts). Next, use an insertion heuristic and Suurballe's algorithm to construct the routing for each ring. Finally, the local search as used in this thesis may then be able to improve the solution by mainly relocating and exchanging SCs between rings. In this way the strengths of both methods are incorporated in one method. The main source of non-optimality will now be in the clustering phase which the local search might be able to overcome partially.

References

Baldacci, R. and Dell'Amico, M. (2010). Heuristic algorithms for the multi-depot ring-star problem. *European Journal of Operational Research*, 203(1):270 – 281.

Baldacci, R., Dell'Amico, M., and González, J. S. (2007). The capacitated m-ring-star problem. *Oper. Res.*, 55(6):1147–1162.

Beasley, J. E. and Nascimento, E. (1996). The vehicle routing-allocation problem: A unifying framework. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 4(1):65–86.

Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90.

Bhandari, R. (1999). *Survivable Networks: Algorithms for Diverse Routing*. Kluwer Academic Publishers.

Brink, R. v. d. (2011). Enabling 4gbb via the last copper drop of a hybrid ftth deployment. TNO White paper 35514. Available online at: `http://www.tno.nl/downloads/WP_Rob_van_den_Brink_Enabling%204GBB%20via%20the%20last%20copper.pdf`.

Campbell, A. M. and Savelsbergh, M. (2004). Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378.

Carroll, P., Fortz, B., Labbé, M., and McGarraghy, S. (2013). A branch-and-cut algorithm for the ring spur assignment problem. *Networks*, 61(2):89–103.

Carroll, P. and McGarraghy, S. (2013). A decomposition algorithm for the ring spur assignment problem. *International Transactions in Operational Research*, 20(1):119–139.

CBS (2012). Statline. bevolking, kerncijfers. Consulted on 8 April 2013 at `http://statline.cbs.nl/StatWeb/publication/?VW=T&DM=SLNL&PA=37296ned&D1=8-13,52-56&D2=0,10,20,30,40,50,(l-1)-l&HD=130408-1459&HDR=G1&STB=T`.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):pp. 568–581.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009). *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, USA, 3rd edition.

Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1995). Chapter 2 time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C. M. and Nemhauser, G., editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 35 – 139. Elsevier.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Eilam-Tzoreff, T. (1998). The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113 – 138.

Fink, A., Schneidereit, G., and Voss, S. (2000). Solving general ring network design problems by meta-heuristics. In Laguna, M. and Velarde, J., editors, *Computing Tools for Modeling, Optimization and Simulation*, volume 12 of *Operations Research/Computer Science Interfaces Series*, pages 91–113. Springer US.

Ford, L. R. (1956). Network flow theory. *The RAND Corporation Paper*, 923.

Fortune, S., Hopcroft, J., and Wyllie, J. (1980). The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111 – 121.

Fouilhoux, P. and Questel, A. (2012). The non-disjoint m-ring-star problem: Polyhedral results and sdh/sonet network design. In Mahjoub, A., Markakis, V., Milis, I., and Paschos, V., editors, *Combinatorial Optimization*, volume 7422 of *Lecture Notes in Computer Science*, pages 93–104. Springer Berlin Heidelberg.

FttH Platform Nederland (2012). Glasmonitor 2012. Available online at: `http://www.ftthplatform.nl/wp-content/uploads/2012/06/291101-Glasmonitor2012_105x210_04-defdef.pdf`.

Gendreau, M., Labbé, M., and Laporte, G. (1995). Efficient heuristics for the design of ring networks. *Telecommunication Systems*, 4(1):177–188.

Gleich, D. (2009). gaimc : Graph algorithms in matlab code. Matlab Central File Exchange, Stanford University. Available online at: `http://www.mathworks.nl/matlabcentral/fileexchange/24134-gaimc-graph-algorithms-in-matlab-code`.

Henningsson, M., Holmberg, K., and Yuan, D. (2006). Ring network design. In Resende, M. G. and Pardalos, P. M., editors, *Handbook of Optimization in Telecommunications*, pages 291–311. Springer US.

Holzer, M., Schulz, F., Wagner, D., and Willhalm, T. (2005). Combining speed-up techniques for shortest-path computations. *ACM Journal of Experimental Algorithmics*, 10.

Hoshino, E. and de Souza, C. (2012). A branch-and-cut-and-price approach for the capacitated m-ring-star problem. *Discrete Applied Mathematics*, 160:2728 – 2741.

Kadaster (2013). Street data amsterdam, the netherlands.

Kalsch, M., Koerkel, M., and Nitsch, R. (2012). Embedding ring structures in large fiber networks. In *Telecommunications Network Strategy and Planning Symposium (NETWORKS), 2012 XVth International*, pages 1–6.

Karp, R. M. (1972). *Reducibility among combinatorial problems*. Springer.

Kindervater, G. and Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. In Aarts, E. and Lenstra, J., editors, *Local search in combinatorial optimization*, section 10, pages 337–360. Wiley.

Kobayashi, Y. and Sommer, C. (2010). On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7(4):234 – 245.

Korte, B., Lovasz, L., Promel, H. J., and Schrijver, A. (1990). *Paths, Flows, and VLSI-Layout*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Labbé, M., Laporte, G., Martín, I. R., José, J., and González, J. S. (2004). The ring star problem: polyhedral analysis and exact algorithm. *Networks*, 43:177–189.

Laporte, G. and Martín, I. R. (2007). Locating a cycle in a transportation or a telecommunications network. *Netw.*, 50(1):92–108.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

Mason, A. (2010). Conceptual approach for the fixed and mobile bulric models. OPTA.

Mauttone, A., Nesmachnow, S., Oivera, A., and Amoza, F. (2008). Solving a ring star problem generalization. In *Conference proceedings for International Conference on Computational Intelligence for Modelling Control and Automation, 2008*, pages 981–986.

Naji-Azimi, Z., Salari, M., and Toth, P. (2010). A heuristic procedure for the capacitated m-ring-star problem. *European Journal of Operational Research*, 207(3):1227–1234.

Naji-Azimi, Z., Salari, M., and Toth, P. (2012). An integer linear programming based heuristic for the capacitated m-ring-star problem. *European Journal of Operational Research*, 217(1):17 – 25.

Phillipson, F. (2013a). A cost effective topology migration path towards fibre. In *The 3rd International Conference on Information Communication and Management (ICICM2013)*, Paris, France.

Phillipson, F. (2013b). Efficient clustering of cabinets at FttCab. In *The 13th International Conference on Next Generation Wired/Wireless Advanced Networking (NEW2AN 2013)*, St.Petersburg, Russia.

Phillipson, F. (2013c). Fast roll-out of fibre-to-the-cabinet: optimal activation of cabinets. Submitted for publication.

Potvin, J. and Rousseau, J. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331 – 340.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Stratix (2013). Glaskaart. Consulted on 5 April 2013 at `http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2010/03/16/next-gen-infrastructures/next-gen-infrastructures-v1-0-3-final.pdf`.

Suurballe, J. W. (1974). Disjoint paths in a network. *Networks*, 4(2):125–145.

Suurballe, J. W. and Tarjan, R. E. (1984). A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336.

TNO (2010). Vraag en aanbod next-generation infrastructures 2010 - 2020. TNO report, The Netherlands. Available online at: `http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2010/03/16/next-gen-infrastructures/next-gen-infrastructures-v1-0-3-final.pdf`.

TNO (2013). Marktrapportage elektronische communicatie december 2012. TNO report, The Netherlands. Available online at: `http://www.tno.nl/content.cfm?context=overtno&content=nieuwsbericht&laag1=37&laag2=2&item_id=2013-02-18%2016:42:09.0`.

Yang, B. and Zheng, S. Q. (2006). Finding min-sum disjoint shortest paths from a single source to all pairs of destinations. In *Proceedings of the Third international conference on Theory and Applications of Models of Computation*, TAMC'06, pages 206–216, Berlin, Heidelberg. Springer-Verlag.

Appendices

A Disjoint shortest paths for two pairs of vertices

The disjoint shortest path problem for two pairs of vertices can be formulated in the following way. Given are two pairs of vertices (s_1, t_1) and (s_2, t_2). Two disjoint shortest paths are sought: one from source s_1 to target t_1 and one from source s_2 to target t_2. The objective is to minimize the sum of the costs of the two paths. In this section two simple methods are described that both fail to find the optimal solution or, even worse, to find a feasible solution when there actually exists one. These two methods and their shortcomings were already discussed shortly in Section 3.3, but here there will be discussed in more detail with some examples.

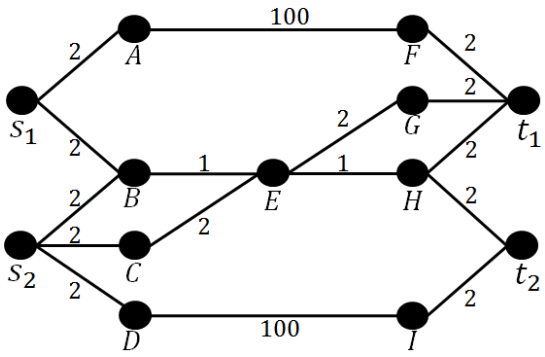
A.1 Deletion method

An intuitive attempt to solve the two pair disjoint shortest paths problem is the following: find the shortest path from source s_1 to target t_1. Next, delete the edges used in this path from the graph and find the shortest path from source s_2 to target t_2 in this subgraph. Then reverse the procedure: find the shortest path from s_2 to t_2 (in the original graph), delete it from the graph and search for the shortest path from s_1 to t_1 in this subgraph. Then choose the cheapest option from the two. As discussed in Section 3.3, there are two main problems with this approach:

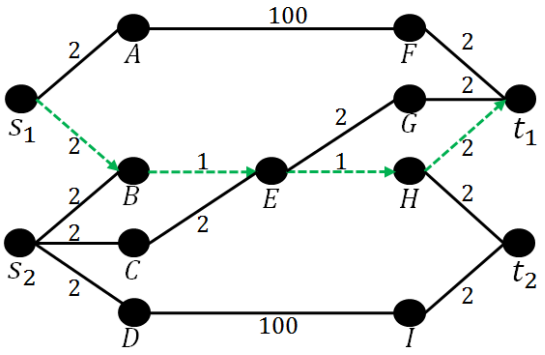
1. The method may fail to find the optimal solution. This is illustrated in the example in Figure 22. In the original graph the shortest path from s_1 to t_1 is found (Figure 22b). Then these edges are deleted from the graph (Figure 22c) and the shortest path from s_2 to t_2 is searched. As one can observe in Figure 22d the expensive edge DI is used, leading to a total cost of 110. The optimal solution are the paths s_1BEGt_1 and s_2CEHt_2, which only have a total cost of 14! It does not help to first look for the shortest path from s_2 to t_2, delete this path, and then search for the shortest path from s_1 to t_1; the same problem remains.

2. Even more worrying is that the method can fail to find a feasible solution, although there actually is one. In Figure 23 such an example is given. It is the same example as in Figure 22, with some edges removed. The optimal solution is still the same. As one can observe from Figure 23c no path from s_2 to t_2 can be found. Also in this case it does not help to change the order of the pairs.

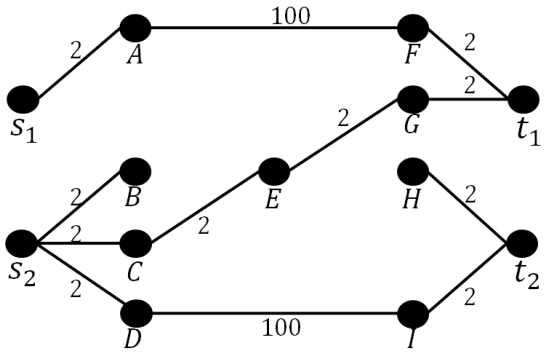
Of course, one can also come up with many examples in which the deletion method does find the optimal solution. For instance, in the example of Figure 25 that is used to show that the method of Bhandari does not work, it does find the optimal solution if one starts with the pair (s_2, t_2). In general, this method and the one discussed next can fail and it is not correct to assume that when the one fails the other works fine (as one may get the impression from the given examples).



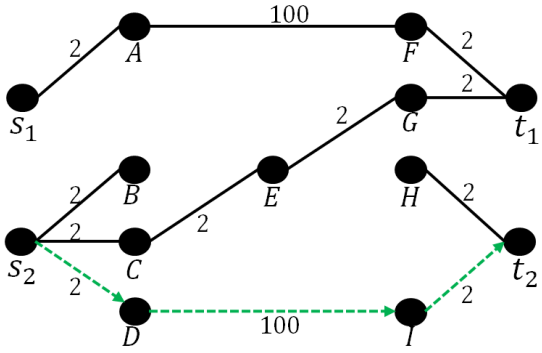
(a) The original graph.



(b) Shortest path from s_1 to t_1 in original graph.

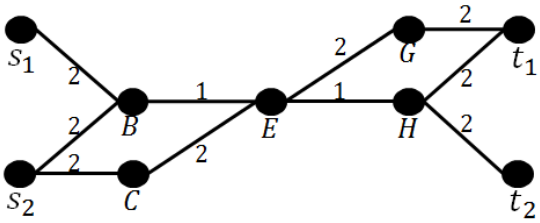


(c) New graph with edges shortest path from s_1 to t_1 deleted.

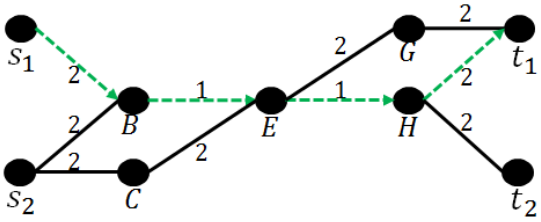


(d) Shortest path from s_2 to t_2 in new graph.

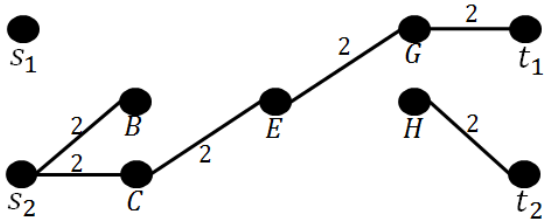
Figure 22: An example with two pairs of vertices for which the deletion method does not find the optimal solution.



(a) The original graph.



(b) Shortest path from s_1 to t_1 in original graph.



(c) New graph with edges shortest path from s_1 to t_1 deleted.

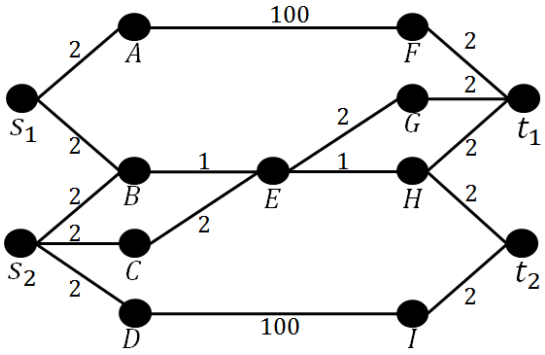
Figure 23: An example with two pairs of vertices for which the deletion method does not find a feasible solution.

A.2 Bhandari method

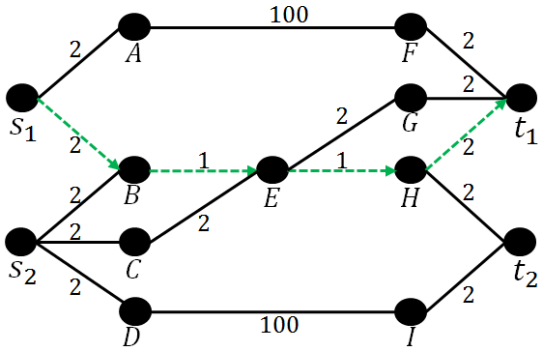
A method due to Bhandari (1999) that works for determining two edge disjoint shortest paths between a single source and a single target is described in Section 3.3. The extension of this method to the more general case of two pairs of vertices is, to the best knowledge of the author, not reported in the literature to fail. Therefore, in this section it is investigated whether application is possible, either direct or by some simple extensions. The result of this investigation is that for the example of Figure 22 in which the 'deletion method' failed it does find the optimal solution. Unfortunately, the method does not always work as is shown by a counter example.

First, an example is given for which the method from Bhandari does give the optimal answer. In Figure 24 the different steps of the method are illustrated for this example. Given is the graph in Figure 24a and goal is to find edge disjoint shortest paths from s_1 to t_1 and from s_2 to t_2 for which the sum of the costs of the two paths is minimal. First the shortest path from s_1 to t_1 is found using Dijkstra: s_1BEHt_1 (see Figure 24b). Then the edges in this path are reversed with respect to the direction they were used. The costs of these reverse arcs are minus the original cost. This gives the graph in Figure 24c. In this graph the shortest path from s_2 to t_2 is found using a shortest path algorithm that can deal with negative weights (e.g. Bellman Ford or 'the modified Dijkstra algorithm' by Bhandari (1999)). This gives the path: $s_2CEGt_1Ht_2$ (Figure 24d). Now edges that are not used in one of the two paths are deleted from the original graph. Moreover, all 'reversed arcs' used in the second path are removed from the original graph as well. This gives the graph in Figure 24e. In this graph the two edge disjoint paths can be found: s_1BEGt_1 and s_2CEHt_2.

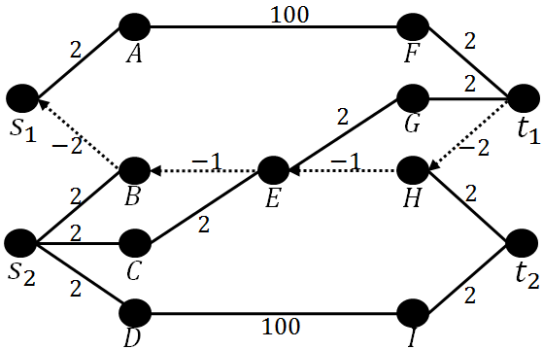
Note that this method fails when it is applied to the graph where e.g. the source s_1 is interchanged with t_1. In other words, in terms of the original graph from Figure 24a two edge disjoint shortest paths are sought: from t_1 to s_1 and from s_2 to t_2. The problem is that then in the graph of Figure 24b (instead of the graph of Figure 24c) a shortest path needs to be found from s_2 to t_2. This shortest path is s_2BEHt_2 resulting in a final graph where, unlike in Figure 24e, the sources are not connected to the sinks. This problem can be resolved in the following way: for one of the pairs, e.g. (s_1, t_1), try both s_1 to t_1 and t_1 to s_1. In this way one of them gives a connected graph as in Figure 24e. In general this method does, however, still not work as is shown in Figure 25. The optimal solution is s_1ACt_1 and s_2ADt_2, but this solution cannot be found with the Bhandari method. In Figure 25e the sources are not connected to the corresponding targets. In this example it does not help to switch a source with a target; the same problem remains. Another idea is to start with the shortest path from s_2 to t_2, reverse the arcs and then find the shortest path from s_1 to t_1. Unfortunately, this gives the same problems.



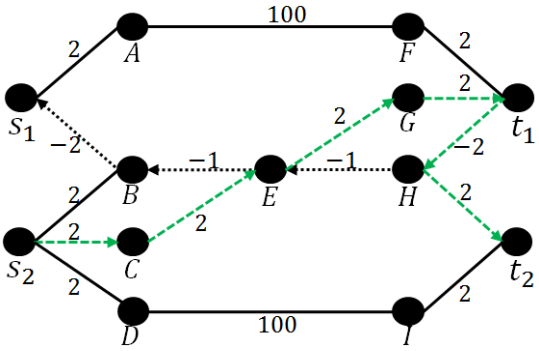
(a) The original graph.



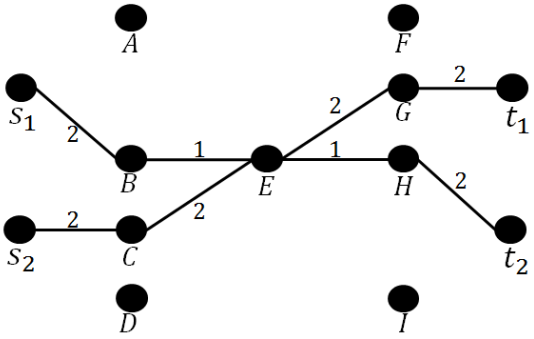
(b) Shortest path from s_1 to t_1 in original graph.



(c) New graph with reversed arcs.

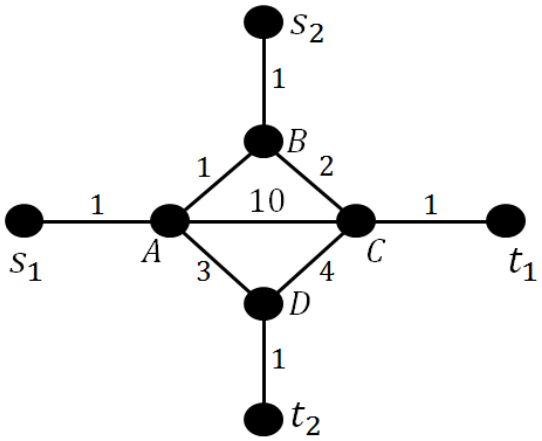


(d) Shortest path from s_2 to t_2 in new graph

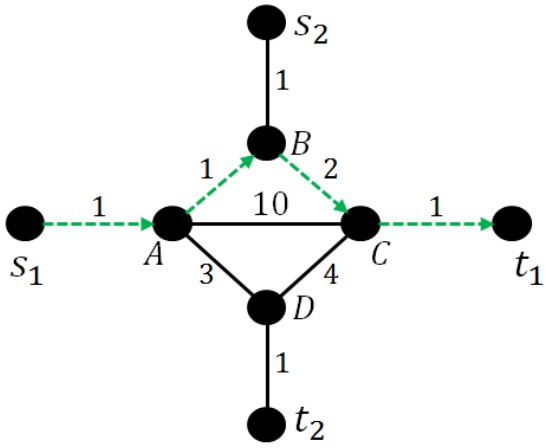


(e) Resulting graph in which the shortest paths can be found.

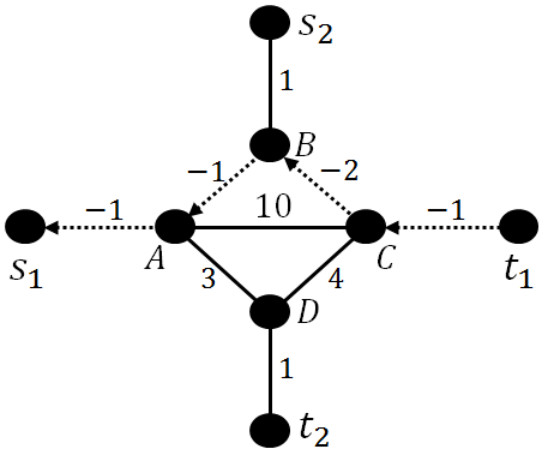
Figure 24: An example with two pairs of vertices for which the method of Bhandari does work.



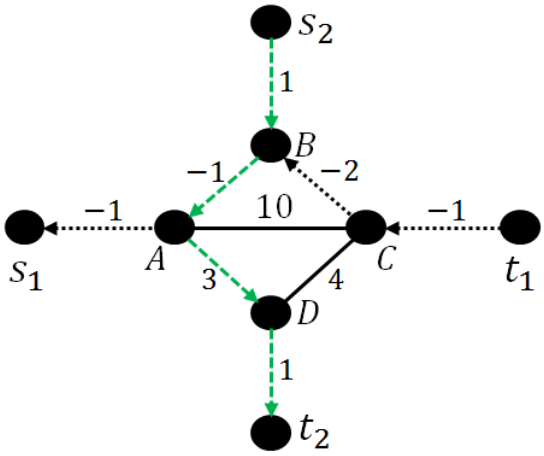
(a) The original graph of the counter example.



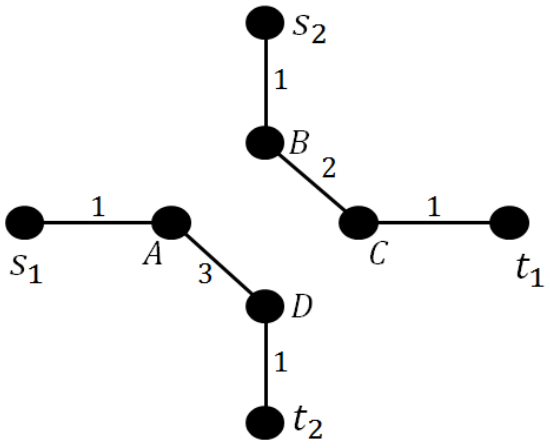
(b) Shortest path from s_1 to t_1 in original graph.



(c) New graph with reversed arcs.



(d) Shortest path from s_2 to t_2 in new graph.



(e) Resulting graph in which the sources and sinks are not connected.

Figure 25: A counter example with two pairs of vertices for which the Bhandari method fails.

B Tables test results

In this Appendix the tables with the test results are given that are described in Section 7. Three different computers were used, since otherwise the testing would have taken too long. They varied a little in their speed and therefore in each table the used computer is mentioned. The specifications of the different computers are:

1. PC A: Intel Core i5 CPU M430 2.27Ghz; 6GB RAM; NVIDIA GeForce GT 320M 3.7GB.
2. PC B: Intel Core i3 CPU M330 2.13GHz; 4GB RAM; Intel HD Graphics 1.6GB.
3. PC C: Intel Core i5-2520M CPU 2.50GHz; 4GB RAM; Intel HD Graphics 1.6GB.

B.1 Test results initialization insertion heuristic

Table 2: Test results seed selection method for a grid of size 10 by 10 with 15 SCs for various duct patterns.

Random instances on a 10 by 10 grid with 15 SCs. Results in this table are based on 1000 replications.
Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 90%, lengthedge=U(20,100). Computer A.

Duct Pattern	Random			Centroid with CO			Centroid without CO			Most spread		
	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)
10 cheap paths	24%	1,700	0.14	29%	1,686	0.15	9%	1,675	0.15	39%	1,669	0.13
5 cheap paths	19%	2,209	0.11	31%	2,167	0.13	9%	2,153	0.12	42%	2,132	0.10
0 cheap paths	20%	2,953	0.04	31%	2,894	0.06	10%	2,867	0.06	39%	2,844	0.04
CO to each SC	22%	2,210	0.09	30%	2,167	0.11	10%	2,161	0.11	38%	2,148	0.09

Table 3: Test results seed selection method for a grid of size 30 by 30 with 50 SCs for various duct patterns.

Random instances on a 30 by 30 grid with 50 SCs. Results in this table are based on 1000 replications.
Other parameters: cap=8, cust=1, cost reduction pipes: 90%, lengthedge=U(20,100). Computer A.

Duct Pattern	Random			Centroid with CO			Centroid without CO			Most spread		
	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)
30 cheap paths	14%	9,038	3.54	19%	8,902	3.73	30%	8,769	3.71	38%	8,745	3.32
15 cheap paths	12%	11,775	3.33	19%	11,505	3.45	30%	11,350	3.39	40%	11,305	3.07
0 cheap paths	7%	20,113	0.84	19%	19,411	1.15	24%	19,241	1.15	51%	18,755	0.84
CO to each SC	12%	7,615	6.32	29%	7,244	6.50	42%	7,136	6.58	18%	7,535	6.48

Table 4: Test results seed selection method for a grid of size 50 by 50 with 100 SCs for various duct patterns.

Random instances on a 50 by 50 grid with 100 SCs. Results in this table are based on 100 replications.
Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 90%, lengthedge=U(20,100). Computer A.

Duct Pattern	Random			Centroid with CO			Centroid without CO			Most spread		
	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)
50 cheap paths	7%	21,785	14.16	17%	21,521	15.80	20%	21,408	15.47	57%	20,871	14.11
25 cheap paths	10%	28,284	15.20	18%	27,848	16.53	27%	27,715	15.65	46%	27,229	14.62
0 cheap paths	6%	56,314	4.07	28%	54,511	6.53	27%	54,486	6.48	40%	53,849	4.11
CO to each SC	11%	18,942	27.66	27%	18,282	27.80	33%	18,073	27.17	29%	18,383	28.94

Table 5: Test results seed selection method for a grid of size 75 by 75 with 300 SCs for various duct patterns.

Random instances on a 75 by 75 grid with 300 SCs. Results in this table are based on 100 replications.
Other parameters: cap=8, cust=1, cost reduction pipes: 90%, lengthedge=U(20,100). Computer A.

Duct Pattern	Random			Centroid with CO			Centroid without CO			Most spread		
	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)	% best	cost	time (s)
75 cheap paths	7%	70,525	104.58	11%	69,792	148.86	32%	69,022	147.50	50%	68,347	113.75
40 cheap paths	8%	86,140	126.08	19%	85,279	169.50	30%	84,583	164.25	43%	84,471	132.70
0 cheap paths	6%	206,289	36.41	49%	199,757	85.04	28%	201,459	83.78	17%	203,108	36.65
CO to each SC	7%	55,012	194.16	19%	53,408	226.35	28%	53,102	228.39	46%	52,823	219.94

B.2 Test results insertion order

Table 6: Test results insertion order for a grid of size 25 by 25 with 60 SCs for various duct patterns.

Random instances on a 25 by 25 grid with 60 SCs. 100 replications. Computer B.
Other parameters: cap=2000, cust=DU(100,200,300,400,500), cost reduction pipes: 80%, lengthedge=U(20,100).

Duct Pattern	**Random order**			**Non-disjoint cost order**			**Disjoint cost order**		
	% best	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**
25 cheap paths	23%	9,954	1.78	26%	9,746	1.89	51%	9,576	34.00
13 cheap paths	9%	12,214	1.77	27%	11,927	1.63	64%	11,656	33.47
0 cheap paths	19%	17,420	0.86	46%	16,891	0.90	35%	16,830	22.28
CO to each SC	17%	8,487	3.26	23%	8,201	3.70	60%	7,919	50.41

Table 7: Test results insertion order for a grid of size 50 by 50 with 100 SCs for various duct patterns.

Random instances on a 50 by 50 grid with 100 SCs. 100 replications. Computer B.
Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 70%, lengthedge=U(20,100).

Duct Pattern	**Random order**			**Non-disjoint cost order**			**Disjoint cost order**		
	% best	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**
50 cheap paths	19%	30,749	6.10	28%	30,253	6.72	53%	29,910	197.28
30 cheap paths	18%	34,622	6.85	26%	34,098	7.88	56%	33,748	219.02
13 cheap paths	21%	42,266	5.92	28%	41,578	6.78	51%	40,835	195.02
CO to each SC	14%	28,033	8.71	20%	27,972	10.13	66%	26,711	260.80

Table 8: Test results insertion order for a grid of size 75 by 75 with 300 SCs for various duct patterns.

Random instances on a 75 by 75 grid with 300 SCs. 25 replications. Computer C.
Other parameters: cap=8, cust=1, cost reduction pipes: 90%, lengthedge=U(20,100).

Duct Pattern	**Random order**			**Non-disjoint cost order**			**Disjoint cost order**		
	% best	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**
75 cheap paths	0%	54,252	128.12	8%	53,111	153.55	92%	50,421	3492.98
40 cheap paths	4%	84,454	70.18	4%	83,761	88.52	92%	80,326	2602.27
0 cheap paths	0%	228,577	20.78	36%	203,737	33.96	64%	202,778	2063.91
CO to each SC	8%	54,002	134.93	3%	53,156	163.61	89%	50,492	3762.66

B.3 Test results extra rings

Table 9: Test results extra rings for a grid of size 40 by 40 grid with 80 SCs.

Random instances on a 40 by 40 grid with 80 SCs, 20 random inexpensive paths, cust=DU(100,200,300,400,500), cap=2500, cost reduction pipes: 60%. 20 replications. Max time local search: 2 min.				
extra rings	**% best**	**cost**	**% best impr**	**cost impr**
0	45%	26,336	45%	23,982
1	45%	26,119	35%	24,235
2	10%	26,389	10%	24,570
3	0%	27,103	5%	25,009
4	0%	27,698	5%	25,359
Use 8.95 % extra rings (approximately 1 extra ring).				

Table 10: Test results extra rings for a grid of size 50 by 50 grid with 150 SCs.

Random instances on a 50 by 50 grid with 150 SCs, 50 random inexpensive paths, cust=DU(100,150,200,300,400), cap=3000, cost reduction pipes: 90%. 40 replications. Max time local search: 2 min.				
extra rings	**% best**	**cost**	**% best impr**	**cost impr**
0	10%	27,929	20%	26,987
1	28%	27,694	33%	26,800
2	23%	27,595	20%	26,807
3	30%	27,501	23%	26,794
4	10%	27,590	5%	26,825
Use 20.33 % extra rings (approximately 2 extra rings).				

Table 11: Test results extra rings for a grid of size 65 by 65 grid with 200 SCs.

Random instances on a 65 by 65 grid with 200 SCs, 60 random inexpensive paths, cust=DU(100,150,200,300,400), cap=2500, cost reduction pipes: 90%. 20 replications. Max time local search: 2 min.				
extra rings	**% best**	**cost**	**% best impr**	**cost impr**
0	20%	40,435	15%	39,439
1	10%	40,100	25%	39,175
2	20%	40,059	20%	39,236
3	40%	39,930	25%	39,120
4	10%	40,041	15%	39,158
Use 15.49 % extra rings (approximately 2 extra rings).				

Table 12: Test results extra rings for a grid of size 75 by 75 grid with 300 SCs.

Random instances on a 75 by 75 grid with 300 SCs, 40 random inexpensive paths, cust=DU(100,200,300,400,500), cap=2500, cost reduction pipes: 75%. 20 replications. Max time local search: 3 min.				
extra rings	**% best**	**cost**	**% best impr**	**cost impr**
0	0%	109,354	5%	104,553
1	5%	108,290	10%	103,768
2	20%	107,034	20%	103,284
3	15%	106,141	20%	103,205
4	10%	106,407	15%	103,373
5	20%	106,194	5%	103,741
6	10%	106,527	15%	103,893
7	5%	107,061	0%	103,999
8	5%	106,992	10%	104,260
9	10%	107,535	0%	105,249
Use 13.61 % extra rings (approximately 5 extra rings).				

Table 13: Test results extra rings for a grid of size 80 by 80 grid with 400 SCs.

Random instances on a 80 by 80 grid with 400 SCs, 40 random inexpensive paths, cust=DU(100,200,300,400,500), cap=2500, cost reduction pipes: 85%. 20 replications. Max time local search: 5 min.				
extra rings	**% best**	**cost**	**% best impr**	**cost impr**
0	0%	127,926	0%	123,417
1	0%	126,740	0%	122,684
2	5%	125,391	0%	121,527
3	20%	124,187	25%	121,064
4	10%	123,954	5%	120,945
5	20%	123,315	35%	120,269
6	5%	123,642	5%	120,753
7	10%	123,438	10%	121,061
8	5%	123,353	0%	121,047
9	15%	123,432	10%	121,378
10	5%	123,905	5%	121,705
11	5%	124,149	5%	122,022
Use 12.07 % extra rings (approximately 6 extra rings).				

Table 14: Test results extra rings for a grid of size 90 by 90 grid with 400 SCs.

Random instances on a 90 by 90 grid with 400 SCs, 60 random inexpensive paths, cust=DU(100,150,200,300,400), cap=2000, cost reduction pipes: 90%. 40 replications. Max time local search: 3 min.				
extra rings	**% best**	**cost**	**% best impr**	**cost impr**
0	3%	111,337	5%	109,079
1	3%	110,528	3%	108,891
2	8%	109,833	8%	108,087
3	8%	109,487	10%	108,111
4	10%	108,813	13%	107,535
5	8%	108,715	15%	107,474
6	5%	108,578	3%	107,194
7	18%	108,510	10%	107,347
8	10%	108,563	13%	107,414
9	10%	108,467	8%	107,350
10	8%	108,556	5%	107,402
11	13%	108,344	10%	107,293
Use 14.18 % extra rings (approximately 6 extra rings).				

Table 15: Test results extra rings for a grid of size 80 by 80 grid with 400 SCs with long improvement time (100 min).

Random instances on a 80 by 80 grid with 400 SCs, 40 random inexpensive paths, cust=DU(100,200,300,400,500), cap=2500, cost reduction pipes: 85%. 2 replications. Max time local search: 100 min.				
	Replication 1		**Replication 2**	
extra rings	**cost**	**cost impr**	**cost**	**cost impr**
0	119,008	109,691	132,815	**120,553**
1	116,681	**108,064**	130,953	122,095
2	115,945	109,095	**129,555**	121,337
3	117,760	109,331	130,151	122,396
4	**114,731**	110,273	131,658	123,453
5	122,066	111,917	131,272	122,608
6	119,177	110,703	131,154	125,220
7	120,813	109,760	130,663	123,937
8	119,816	110,885	131,181	124,205
9	119,112	109,875	130,715	124,284
10	117,734	109,345	130,607	124,801
11	118,121	110,037	131,507	124,957
Based on initial cost: use 6.26 % extra rings (approximately 3 extra rings). Based on cost impr: use 1.02 % extra rings (approximately 0 extra rings).				

B.4 Test results comparison insertion heuristic and cluster first - route second heuristic

Table 16: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 10 by 10 with 15 SCs for various duct patterns.

Random instances on a 10 by 10 grid with 15 SCs. Results in this table are based on 1000 replications. Max time local search: 5 min. Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 90%, lengthedge=U(20,100). Computer B.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
10 cheap paths	77%	1,670	0.15	65%	1,548	0.98	23%	1,905	0.10	35%	1,608	1.62	4%
5 cheap paths	71%	2,138	0.11	60%	1,988	1.33	29%	2,315	0.10	40%	2,042	1.87	3%
0 cheap paths	53%	2,861	0.06	53%	2,717	0.28	47%	2,883	0.09	47%	2,734	0.30	3%
CO to each SC	91%	1,356	0.27	70%	1,248	1.47	9%	1,828	0.11	30%	1,338	1.86	5%

Table 17: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 20 by 20 with 50 SCs for various duct patterns.

Random instances on a 20 by 20 grid with 50 SCs. Results in this table are based on 100 replications. Max time local search: 5 min. Other parameters: cap=8, cust=1, cost reduction pipes: 50%, lengthedge=U(20,100). Computer B.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
20 cheap paths	58%	10,073	0.70	59%	9,217	80.76	42%	10,302	0.73	41%	9,288	101.30	2%
10 cheap paths	56%	11,062	0.66	56%	10,117	74.99	44%	11,164	0.73	44%	10,194	98.03	2%
0 cheap paths	58%	12,841	0.59	58%	11,773	61.67	42%	12,650	0.77	42%	11,873	72.84	3%
CO to each SC	62%	9,025	1.19	62%	8,082	83.09	38%	9,273	0.75	38%	8,210	129.11	2%

Table 18: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 30 by 30 with 100 SCs for various duct patterns.

Random instances on a 30 by 30 grid with 100 SCs. Results in this table are based on 100 replications. Max time local search: 5 min. Other parameters: cap=2500, cust=DU(100,200,250,300,400), cost reduction pipes: 80%, lengthedge=U(20,100). Computer A.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
30 cheap paths	93%	16,716	4.88	83%	15,375	301.50	7%	18,339	3.01	18%	15,957	302.23	5%
15 cheap paths	82%	20,285	4.83	72%	18,489	301.42	18%	21,485	2.92	28%	19,116	300.97	5%
0 cheap paths	39%	30,116	2.21	44%	27,134	289.88	61%	29,854	2.70	56%	27,081	300.41	3%
CO to each SC	96%	13,730	8.51	93%	12,088	302.61	4%	15,685	3.12	7%	13,394	302.86	5%

Table 19: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 50 by 50 with 100 SCs for various duct patterns.

Random instances on a 50 by 50 grid with 100 SCs. Results in this table are based on 100 replications. Max time local search: 5 min. Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 60%, lengthedge=U(20,100). Computer C.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
50 cheap paths	61%	34,370	3.74	63%	31,345	301.25	39%	34,827	2.61	37%	31,794	301.69	0%
25 cheap paths	64%	37,979	3.78	62%	34,706	301.12	36%	38,660	2.69	38%	35,084	300.79	0%
0 cheap paths	34%	53,688	2.61	45%	48,087	300.91	66%	52,312	2.44	55%	47,744	300.91	0%
CO to each SC	57%	32,458	4.99	72%	28,443	301.43	43%	32,694	2.78	28%	29,326	302.15	1%

Table 20: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 60 by 60 with 50 SCs for various duct patterns.

Random instances on a 60 by 60 grid with 50 SCs. Results in this table are based on 100 replications. Max time local search: 5 min.
Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 85%, lengthedge=U(20,100). Computer B.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
45 cheap paths	67%	16,451	5.94	70%	15,134	295.82	33%	16,836	2.73	30%	15,434	297.94	0%
20 cheap paths	57%	21,667	5.94	68%	19,619	299.36	43%	21,947	2.56	32%	20,044	300.06	0%
10 cheap paths	44%	26,298	5.41	58%	23,563	288.70	56%	26,031	2.53	42%	23,806	294.60	0%

Table 21: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 70 by 70 with 250 SCs for various duct patterns.

Random instances on a 70 by 70 grid with 250 SCs. Results in this table are based on 100 replications. Max time local search: 5 min.
Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 75%, lengthedge=U(20,100). Computer A.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
60 cheap paths	100%	80,231	48.79	100%	77,829	309.82	0%	90,811	35.19	0%	86,764	305.17	1%
30 cheap paths	100%	93,156	53.96	100%	90,243	310.75	0%	107,376	34.68	0%	102,596	307.60	3%
10 cheap paths	100%	124,093	51.88	100%	120,282	308.32	0%	137,148	34.69	0%	132,483	304.08	1%
CO to each SC	100%	69,208	67.34	100%	65,972	311.72	0%	82,791	35.35	0%	77,721	306.09	1%

Table 22: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 85 by 85 with 350 SCs for various duct patterns.

Random instances on a 85 by 85 grid with 350 SCs. Results in this table are based on 100 replications. Max time local search: 5 min.
Other parameters: cap=12, cust=1, cost reduction pipes: 80%, lengthedge=U(20,100). Computer A.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
75 cheap paths	74%	92,419	90.71	80%	91,347	333.78	26%	94,459	106.19	20%	93,830	353.62	2%
40 cheap paths	74%	105,648	99.25	79%	104,396	337.02	26%	107,759	106.39	21%	107,139	351.36	0%
CO to each SC	100%	75,345	168.80	100%	74,617	375.75	0%	83,359	107.48	0%	82,582	367.73	0%

Table 23: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 95 by 95 with 500 SCs for various duct patterns.

Random instances on a 95 by 95 grid with 500 SCs. Results in this table are based on 100 replications. Max time local search: 5 min.
Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 85%, lengthedge=U(20,100). Computer A.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
20 cheap paths	100%	216,722	351.89	100%	214,315	343.30	0%	282,130	205.17	0%	276,948	328.37	1%
10 cheap paths	100%	277,191	348.96	100%	274,064	335.13	0%	341,994	204.94	0%	336,696	334.07	1%
0 cheap paths	100%	411,080	139.25	100%	404,760	319.57	0%	452,199	198.81	0%	447,366	319.72	3%

Table 24: Test results of the comparison between the insertion heuristic and the cluster first - route second heuristic for grid of size 100 by 100 with 600 SCs for various duct patterns.

Random instances on a 100 by 100 grid with 600 SCs. Results in this table are based on 100 replications. Max time local search: 5 min.
Other parameters: cap=2500, cust=DU(100,200,300,400,500), cost reduction pipes: 90%, lengthedge=U(20,100). Computer B.

	Insertion heuristic						**Cluster first - route second heuristic**						
	initial solution			**after local search**			**initial solution**			**after local search**			
duct pattern	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time(s)**	**% best**	**cost**	**time (s)**	**% best**	**cost**	**time (s)**	**no sol**
100 cheap paths	100%	148,182	657.84	100%	147,138	364.85	0%	187,148	399.81	0%	184,478	344.91	0%
50 cheap paths	100%	181,754	738.17	100%	180,383	374.91	0%	243,458	393.29	0%	239,713	349.01	0%
25 cheap paths	100%	225,423	727.83	100%	223,771	361.00	0%	309,579	395.22	0%	305,582	346.23	2%

C Illustration of some solutions

In this Appendix some figures are presented of solutions of the different algorithms to give the reader a better insight in the kind of solutions that are obtained. The rings are indicated by different colors in the grids. If rings overlap on an edge, only of one them is shown.

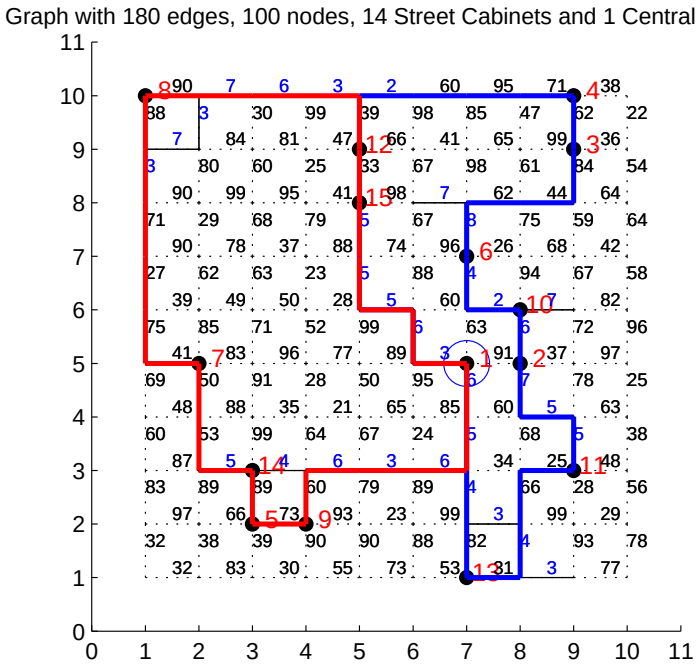


Figure 26: The solution (cost: 1,689) of the insertion heuristic for an instance of size 10 by 10 with 14 SCs, 5 inexpensive paths, cost reduction pipes = 90%, $cap = 2500$, $cust = DU(100, 200, 300, 400, 500)$.

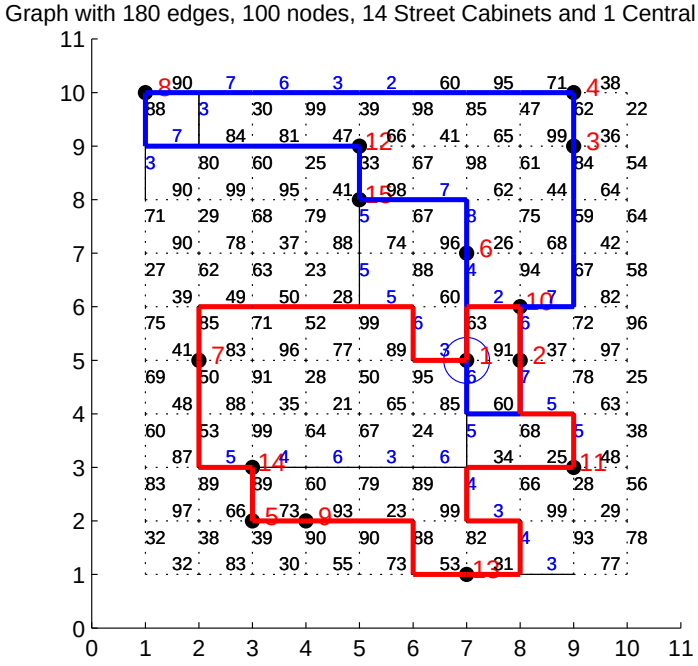


Figure 27: The solution (cost: 2,150) of the cluster first - route second heuristic for the same instance as in Figure 26.

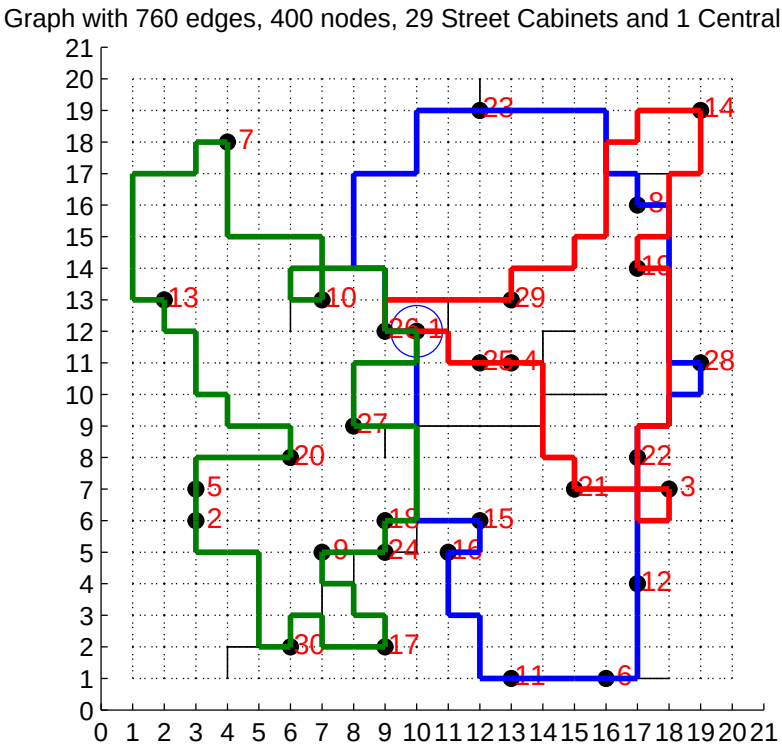


Figure 28: The solution (cost: 6,016) of the insertion heuristic for an instance of size 20 by 20 with 39 SCs, 5 inexpensive paths, cost reduction pipes = 90%, $cap = 2500$, $cust = DU(100, 200, 300, 400, 500)$.

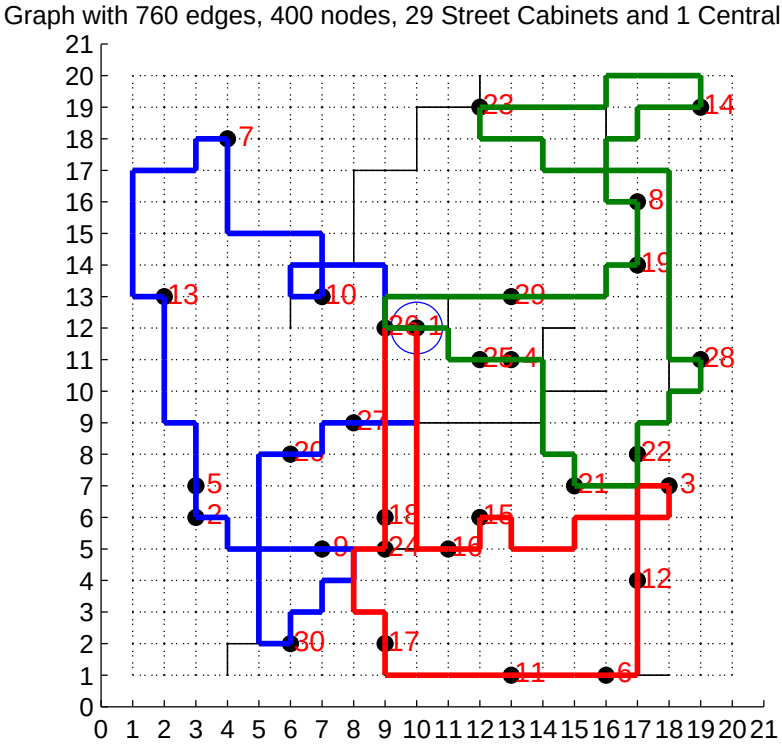


Figure 29: The solution (cost: 6,468) of the cluster first - route second heuristic for the same instance as in Figure 28.

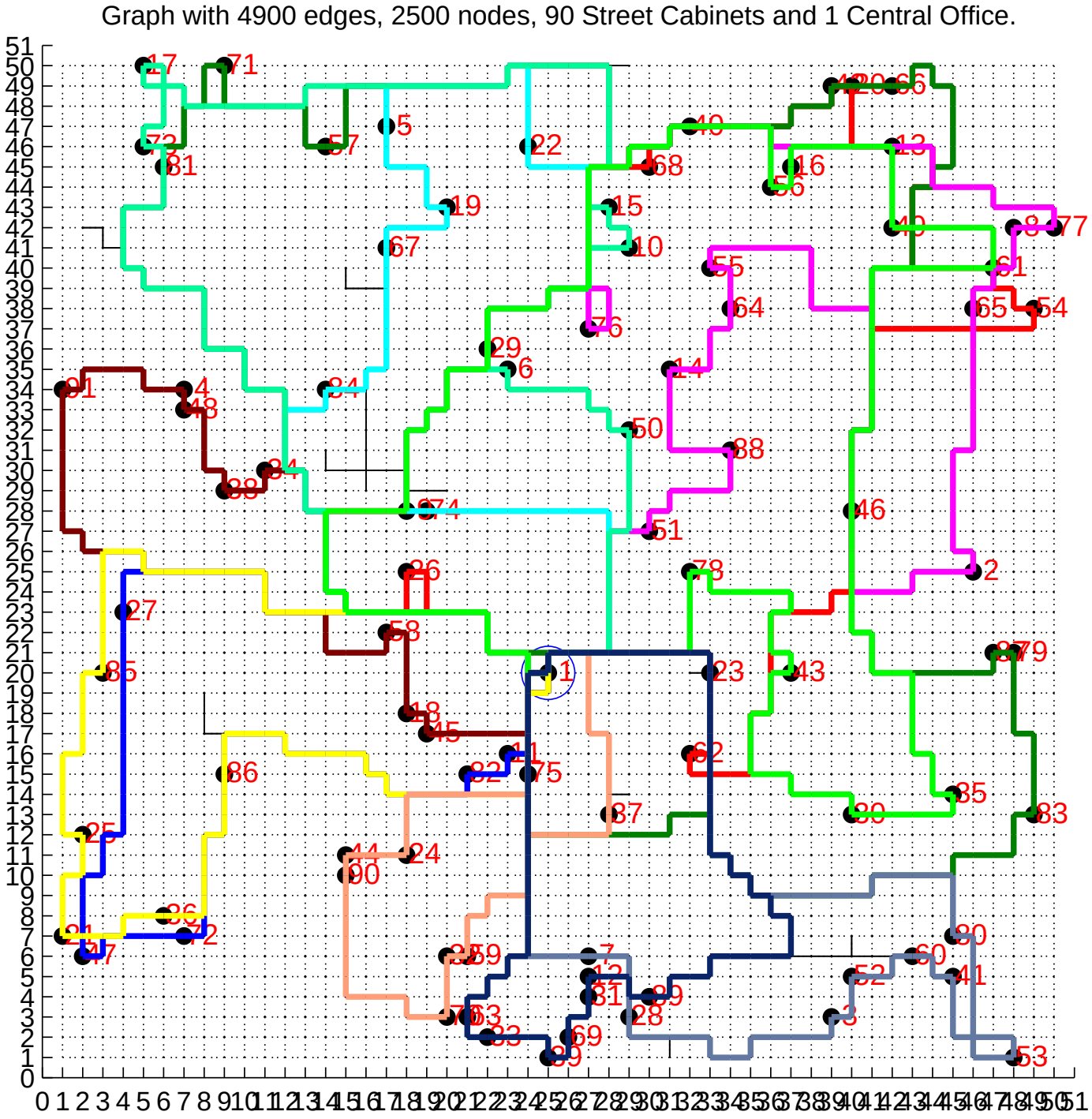


Figure 30: The solution (cost: 31,405) of the insertion heuristic for an instance of size 50 by 50 with 90 SCs, 5 inexpensive paths, cost reduction pipes = 90%, $cap = 2500$, $cust = DU(100, 200, 300, 400, 500)$.

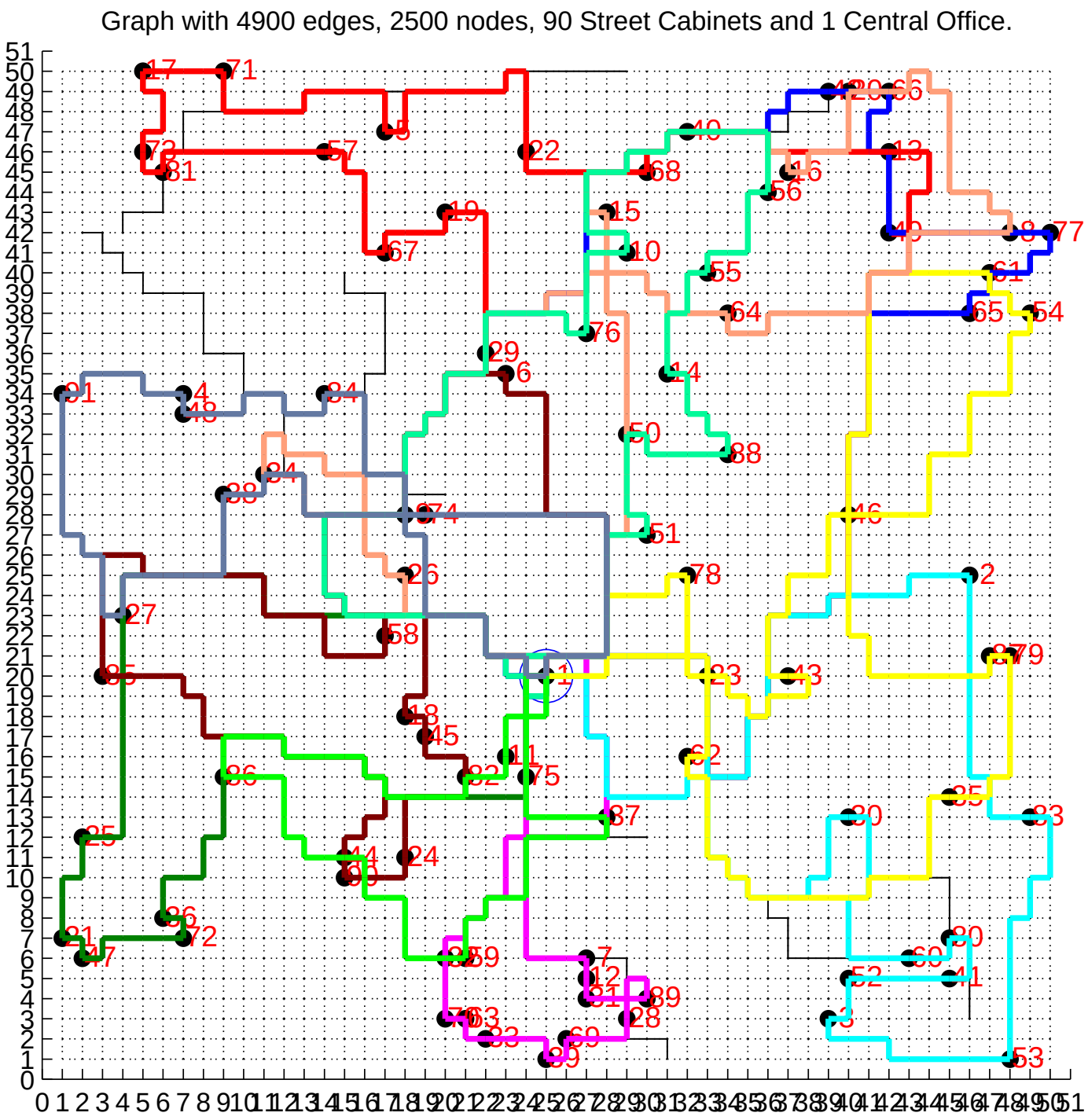


Figure 31: The solution (cost: 35, 186) of the cluster first - route second heuristic for the same instance as in Figure 30.

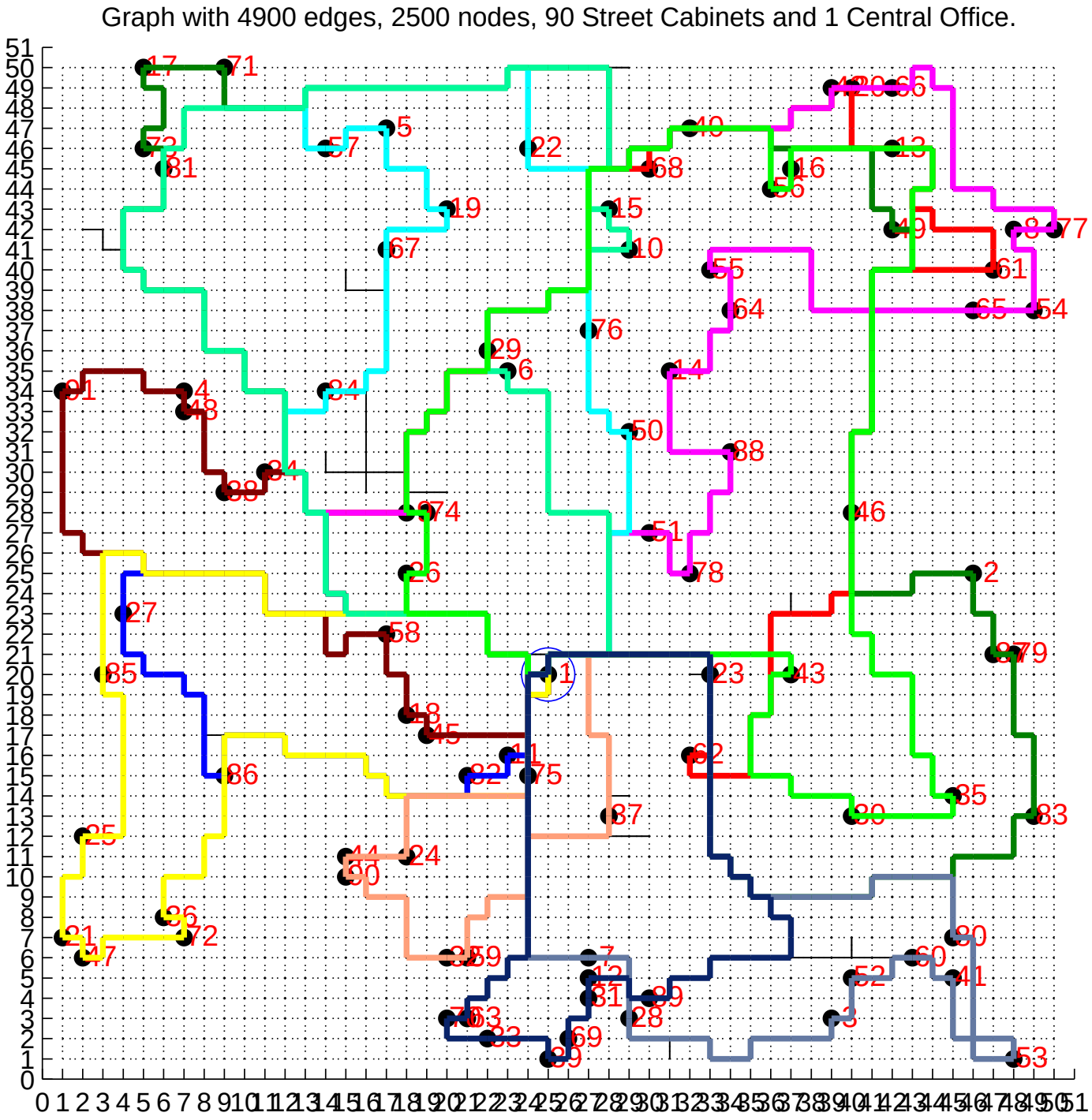


Figure 32: The improved solution (cost: 28,070) of the insertion heuristic for the instance in Figure 30. Time improvement: 32,5 min. 10.62% improvement.