Subway design in Utrecht

Silvia Cuadrado
Universitat Autònoma de Barcelona.

Kim Feldhoff
University of Kaiserslautern.

Rafał Latkowski
University of Warsaw.

Tobias Werther
University of Vienna.

Martijn Zegers
Eindhoven University of Technology.

Problem holder: Job Smeltink
University of Utrecht.

Abstract

This report is dealing with the design of a subway in Utrecht. The main goal is to find a (sub)optimal subway track with respect to the travel times. The problem is translated into a combinatorical optimization problem.

Because of the complexity of the problem we think that the best approach is to use local search methods. For evaluating the quality of the subway one has to calculate the total travel time. We line out the steps for computing or estimating this value. The local search algorithms presented are Hill Climbing, Taboo Search and Simulated Annealing. Furthermore, two possible neighbourhood structures are defined, namely "Adding and deleting stops" and "Exchanging stops". To improve the speed of the search we introduce a preselection of the neighbourhood. Finally we give some conclusions and recommendations.

1 Introduction to the Problem

Utrecht, situated in the center of the Netherlands, wants to add one subway line to the existing network of bus lines. The subway has to pass the central station and replaces several bus stops. The advantage of the design is that travellers are getting faster from one point to another. Since the capacity of a subway can be easily adjusted by removing or adding a compartment, the subway is more flexible in rush hours. On the other hand, the construction of a subway track is very expensive. Therefore, the length of the subway track is limited to 10 km. To ensure, that the subway travels fast enough, the distance of two subway stops should be at least 1 km.

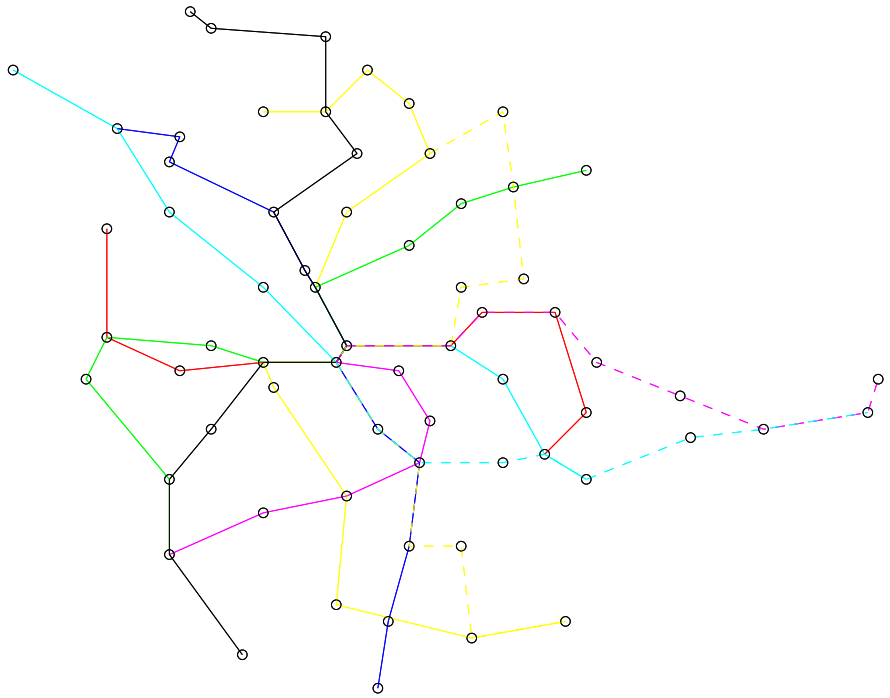

Figure 1: network of bus lines and stops in Utrecht

The aim of this report is to describe the subway design in a mathematical model in which we can state some solution methods for a (sub)optimal subway track.

2 Model

In this section we translate the problem in a mathematical model.

2.1 Given Data

- We have given a network of fixed bus lines with $N = 66$ different bus stops, which can be discribed by the graph

$$G_0 = (V_0, E_0). \tag{1}$$

  $V_0 = \{1, \ldots, N\}$ is the set of bus stops,
  $E_0 = \{e = \{i, j\} : \text{stop } i \text{ and stop } j \text{ are connected directly}\}$.

- The matrix $\mathcal{D} = (d_{ij})$ denotes the distances between the bus stops i and j.

- The matrix $\mathcal{C} = (c_{ij})$ denotes the numbers of passengers travelling from i to j.

- Every stop will be served by a bus and/or a subway in abouth $10\,min$. This implies, that the changing time from one line to another will be on average $5\,min$.

- The maintenance costs of a bus is denoted by m_b and for the subway by m_s.

2.2 The Problem

In order to simplify the problem we will not change the given bus lines and restrict the problem to a fixed network.

Since we want to build a subway line through some given bus stops, the set of vertices remains the same, but because of additional subway connection, the set of edges will increase:

$$G = G_0 \cup G_s = (V, E) = (V_0, E_0) \cup (V_s, E_s) = (V_0, E_0 \cup E_s), \tag{2}$$

where

- The graph G_s is a connected graph.

- $V_s \subseteq V_0$ is the set of all stops belonging to the subway line wich has to satisfy the conditions that:

$$\begin{aligned} deg(v) &\leq 2 \quad \forall_{v \in V_s}, && (3)\\ v_0 &\in V_s, && (4)\\ & && (5) \end{aligned}$$

  where v_0 denotes the central station.

- E_s is the set of all direct connections in the subwayline with the following restrictions:

$$\begin{aligned} \sum_{e \in E_s} d_e &\leq 10, && (6)\\ d_e &\geq 1 \quad \forall_{e \in E_s}. && (7) \end{aligned}$$

We measure the quality of the graph G by the weighted total travel time and the maintenance costs per day of the buses and the subway, denoted by $T(G)$ and $C(G)$, respectively.

(a) The weighted total travel time is the sum of travel times over all trips from i to j weighted by the numbers of passengers travelling from point i to j, i.e.,

$$T(G) = \sum_{i,j} c_{ij} t_{ij} \tag{8}$$

where t_{ij} is the shortest travel time from i to j.

(b) The costs per day consist of the number of buses and subways travelling per day multiplied by the maintenance costs, i.e.,

$$C(G) = \sum_{k=1}^{K} m_b N_{b,k} + m_s N_s \tag{9}$$

where

- K is the number of bus lines.
- $N_{b,k}$ is the number of buses travelling on bus line k.
- N_s is the number of subways travelling on the subway line.

We are interested in finding a balance between the travel times and the costs, since we want to minimize both. The minimize function can be written as

$$f(G) = \alpha T(G) + (1 - \alpha)C(G) \tag{10}$$

where $\alpha \in [0, 1]$.

In the following, we set $\alpha = 1$ since we assume that $C(G)$ is almost constant for any added subway line and because we keep the network of buses fixed.

3 Computation of the Travelling Time

For the total travel time it is important to find the shortest travel time for every pair of points (i, j) in the graph G. This can be done by applying a shortest path algorithm. Before that, we have to modify the graph with respect to the changing stops in order to take into account the changing time. At every point, where passengers can change from one line to another, we will add so many additional "virtual points", that every changing possibility will be covered. They are useful to describe the changing time in form of additional edges τ.

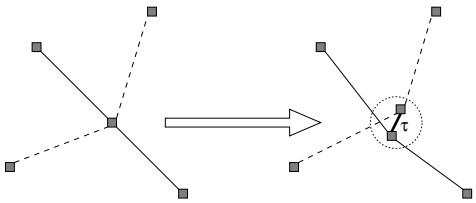

Figure 2: Virtual points replace changing point.

In the given graph G it is possible that there are multiple edges. After modifying the graph there are no multiple edges anymore. A suitable shortest path algorithm for our problem is the Floyd algorithm described as follows.

Initial matrix: $\Phi^0 = (\phi^0_{ij})$:
for $m = 1, \ldots, N + p$
  for $i = 1, \ldots, N + p$
    for $j = 1, \ldots, N + p$
      $\phi^m_{ij} = min\{\phi^{m-1}_{im} + \phi^{m-1}_{mj}, \phi^{m-1}_{ij}\}$
    end.

end.
end.

Φ^0 is a $(N+p) \times (N+p)$ matrix where p is the number of changing possibilities from one line to another in the given network. ϕ^0_{ij} is the travel time on the shortest edge from vertex i to vertex j. If no such edge exists, set $\phi^0_{ij} = \infty$. Moreover, $\phi^0_{ii} = 0$.
The complexity of the Floyd algorithm is $\mathcal{O}\ ((N+p)^3)$.

The resulting matrix $\Phi^{N+p} = T^p$ has now to be reduced again by the number of virtual points. This means that we modify T^p such that we combine the corresponding virtual points to the original one, i.e.,

$$T^p \in \mathcal{R}^{(N+p)\times(N+p)} \longrightarrow T \in \mathcal{R}^{N\times N}. \tag{11}$$

In particular, if $i_1, \ldots, i_k$ are the virtual points for the vertex i, we set

$$t_{ij} = \min_{1\le k\le K}\{t_{i_k j}\},$$
$$t_{li} = \min_{1\le k\le K}\{t_{l i_k}\}.$$

Now, we can compute $T(G) = \sum_{i,j} c_{ij} t_{ij}$ for every graph G. A simple implementation of the algorithm permits to classify different networks in terms of the total travel time.

4 Approach

After the introduction of an implementable cost function, we are able to define a solution method. We have a computable measure — the weight travelling time assigned to the selected subway line. We look for subway lines where the travel time will be small. Hence, the aim is to find a subway line that minimizes our measure. This minimization problem on the graph structure belongs to the well known family of combinatorial optimization problems. This kind of problem does not permit an explicit formulation of an optimal solution. We have to find it in a set of various possible solutions. However, the set of possible solutions is huge, so it is impossible to look at all possible subway lines and evaluate their weighted travel time in an acceptable time. Fortunately, there exist some efficient heuristic methods that can find near-optimal solution. This methods are called **local search methods**.

4.1 Relation of Neighbourhood

Let S be a possible solution space. This means that S consists of all possible subway lines that satisfy the given conditions.

Next, we introduce a relation of neighbourhood. For any $s_1, s_2 \in S$ we shall say that s_1 and s_2 are neighbours in a solution space S if the difference between s_1 and s_2 is small with respect to the defined measure. The structure of the solution space S depends on the definition of the concept of the neighbourhood. Figure 3 shows an example of two neighbours and the corresponding solution space S with neighbourhood structure represented as a graph. The points s_1 and s_2 are connected i.f.f. they are neighbours.

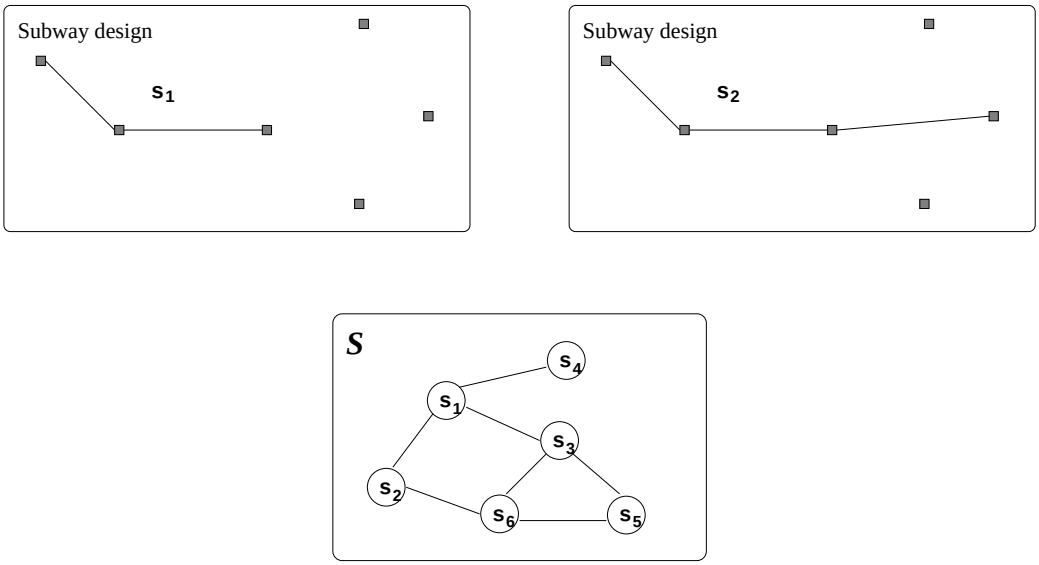

Figure 3: Neighbours and neighbourhood structure of a solution space S.

4.2 The Idea of Local Search

It is very useful to think about the neighbourhood structure of S in the graph representation. All local searches are based on the idea of searching optimal solutions by following these connections. Local search algorithms consider local changes and find in which direction we have to move to approximate an optimal solution in the searching space. Local change means that we exchange a current solution for any of the neighbours, and select the best solution. If the relation of the neighbourhood is introduced in accordance with variability of subway properties, we can expect that local search algorithms will be able to detect an optimal solution.

By comparing neighbours we are able to find near-optimal solutions. However, in most cases we have to compare a current solution with all neighbours. A typical neighbourhood size in our project is about six hundred. This means that we have to compute about 600 measures of subway lines for one step in a local search algorithm. The problem is that local search algorithms needs a lot of iteration steps. The Floyd Shortest Path algorithm is fast for our problem, however we can not afford to apply it 600 times per iteration. This is the main reason why we have developed some implementational improvements in the computation of the weighted travel time. These improvements are described in following section.

5 Measurement of the Subway Lines

In this section we will attach weights to the edges and the subway which will allow us to test several different strategies for the implementation of the subway in the graph we have defined. With these weights we can compute the travel time for every different graph without using the shortest path algorithm every time.

Recall that $T(G_0) = \sum_{ij} c_{ij} t_{ij}$ denotes the total travel time (for the graph G_0) defined in Section 2. When adding edges of the subway, we make the assumption that we do it in a consecutive way, that is, once we fix one edge of the subway $\{a, b\}$, the next edge will be $\{b, c\}$ for some c. Let us assume that we build a subway with only one edge $\{a, b\}$ (that is, we have two stations a and b and one line connecting them). Then the weight that measures the new subway is the total travel time of the new graph, that is, $G_0 \cup \{a, b\}$ (the initial graph with the additional edge $\{a, b\}$). Denoting the weight by ω_{ab}, we have

$$\omega_{ab} = T\big(G_0 \cup a, b\big) = \sum_{ij} c_{ij} t_{ij}^{(ab)}. \tag{12}$$

Changing a and b we have the weights for all possible subways with only one edge. For a subway with two edges (a, b), (b, c) the weight is

$$\omega_{abc} = T\big(G_0 \cup a, b \cup b, c)\big) = \sum_{ij} c_{ij} t_{ij}^{(abc)} \tag{13}$$

and it can be easily seen that

$$t_{ij}^{(abc)} = \min\big(t_{ij}^{(ab)}, t_{ij}^{(bc)}, t_{ij}^{(ac)} + t_{ab} + t_{bc} - t_{ac}\big). \tag{14}$$

The term $t_{ij}^{(ab)}$ means that only the edge a, b is used, the term $t_{ij}^{(bc)}$ means that only the edge b, c is used and the term $t_{ij}^{(ac)} + t_{ab} + t_{bc} - t_{ac}$ means that both edges are used.

We can generalize this formula for a subway with k edges $\{a_1, a_2\}, \ldots,$ $\{a_k, a_{k+1}\}$ in the following way:

$$\begin{aligned} t_{ij}^{(a_1 \ldots a_{k+1})} \;=\; \min \Big(& t_{ij}^{(a_1 a_2)}, t_{ij}^{(a_2 a_3)}, \ldots, t_{ij}^{(a_k a_{k+1})}, \\ & t_{ij}^{(a_1 a_3)} + t_{a_1 a_2} + t_{a_2 a_3} - t_{a_1 a_3}, \ldots, \\ & t_{ij}^{(a_{k-1} a_{k+1})} + t_{a_{k-1} a_k} + t_{a_k a_{k+1}} - t_{a_{k-1} a_{k+1}}, \\ & t_{ij}^{(a_1 a_4)} + t_{a_1 a_2} + t_{a_2 a_3} + t_{a_3 a_4} - t_{a_1 a_4}, \ldots, \\ & t_{ij}^{(a_{k-2} a_{k+1})} + t_{a_{k-2} a_{k-1}} + t_{a_{k-1} a_k} + t_{a_k a_{k+1}} - t_{a_{k-2} a_{k+1}}, \\ & \ldots \\ & t_{ij}^{(a_1 a_k)} + t_{a_1 a_2} + \cdots + t_{a_{k-1} a_k} - t_{a_1 a_k}, \\ & t_{ij}^{(a_2 a_{k+1})} + t_{a_2 a_3} + \cdots + t_{a_k a_{k+1}} - t_{a_2 a_{k+1}}, \\ & t_{ij}^{(a_1 a_{k+1})} + t_{a_1 a_2} + \cdots + t_{a_k a_{k+1}} - t_{a_1 a_{k+1}} \Big) \end{aligned} \tag{15}$$

We have k terms representing passengers that only use one edge of the subway, $k-1$ terms representing passengers that use two edges, $k-2$ terms representing passengers that use three edges and so on, and, finally, one term representing passengers that use the k edges of the subway.
So, in total we have

$$k + (k-1) + (k-2) + \cdots + 1 = \frac{k(k+1)}{2} \tag{16}$$

terms to find the minimum. The complexity of calculating the total weighted travel time is now $\mathcal{O}\left((Nk)^2\right)$ (where N is the number of stops).

6 Local Search Algorithms

In this section we would like to introduce some local search algorithms that are our key to find the (sub)optimal subway design. In the next section we present how we can apply these algorithms to our problem.

6.1 Algorithms

We would like to present three algorithms as our approach to find the optimal subway design.

- Hill Climbing (HC)
- Taboo Search (TS)
- Simulated Annealing (SA)

Each of the algorithms have some advantages and disadvantages, however we can rank these algorithms in the order above. The HC algorithm is the fastest, however it can not avoid local optima to find the global one. In the other hand SA should always find the global optimum, however it can take a lot of time. We have to investigate and try each algorithm for our problem, because efficiency and ability to find near-optimal solutions depend on the complexity of the solution space in terms of density and size of local optima, which can not be examined explicitly.

6.2 Hill Climbing

```
current_solution := randomize( solution_space );
best := current_solution;
repeat
   S := set_of_neighbours( current_solution );
   next := select_best( S );
   if (next is_better_then current_solution)
   then
      current_solution := next
   else
      if (current_solution is_better_then best)
      then best := current_solution;
      current_solution := randomize( solution_space );
until time_out;
if (current_solution is_better_then best)
then best := current_solution;
```

Hill Climbing is a simple algorithm. It starts from the randomized initial state (in our case from the randomized subway design) and looks at each neighbour to find the best one. If the best neighbour is better than the current solution then it replaces the current solution. If the best neighbour is not better then we have stacked in a local optimum, and we have to start

our process again from the other initial state. It means that in this case we have to re-randomize the current solution to start the process in another place of the possible solution space. Also the algorithm stores the best local optimum before re-randomizing the current solution.

This algorithm is very easy to implement and it also runs to each local optimum very quickly. However, it can only find a local optimum. If the solution space is very complex we can never reach a satisfactory local optimum.

6.3 Taboo Search

```
current_solution := randomize( solution_space );
best := current_solution;
create_empty_list( taboo_list , taboo_list_size );
repeat
   add( taboo_list , current_solution );
   S := set_of_neighbours( current_solution );
   S := S - taboo_list;
   current_solution := select_best( S );
   if (current_solution is_better_then best)
   then best := current_solution;
until time_out;
```

This algorithm looks like the previous one (HC). The main difference is that here we introduce very clever mechanism that can help to avoid local optima, however this mechanism is a little bit expensive, so we have to trade off between speed and quality. In this algorithm we have an additional solution list implemented as a cyclic table where we can store the last T solutions. This is our taboo list. The algorithm can not select again the solution from the taboo list, also the next current solution is simply the best solution among the set of neighbours, even though it is not a better solution in comparison to the current one. The quality of the algorithm in terms of the ability of avoiding local optima, is better with an increasing length of the taboo list, however comparing two subway designs is an expensive operation, and this list can not be as long as we wish.

6.4 Simulated Annealing

```
current_solution := randomize( solution_space );
best := current_solution;
temperature := max_temperature;
```

```
while ( temperature > 0 ) do
   repeat
      next := select_randomly_neighbour( current_solution );
      delta := F( next ) - F( current_solution );

      if ( delta < 0 )
      then
         current_solution := next;
      else
         x := randomize_uniform(0,1);
         if ( x < exp( - delta / temperature ) )
         then current_solution := next;

      if (current_solution is_better_then best)
      then best := current_solution;
   until stabilization_criterion( current_solution );
   decrease( temperature );
```

The Simulated Annealing algorithm differs from the two previous algorithms, and can find a global optimum even more frequently than TS can do. However, it is the most expensive algorithm. This algorithm was developed by Metropolis in 1953 and it is proved, that it can avoid local optima in any minimization problem.

In each iteration step we try to make a random change. If this change gives an improvement measured by the quality function (reduction of this value) then we select it as the new current solution. However, even if the changed solution is not so good as the current state there is still some nonzero probability to accept this solution. Let s_c and s_n respectively be a current and next (after change) solution and $F(s)$ be a quality function. The probability of acceptance of change is:

$$P(s_c \to s_n) = \begin{cases} 1 & F(s_n) - F(s_c) \leq 0 \\ e^{-\frac{F(s_n)-F(s_c)}{T}} & F(s_n) - F(s_c) > 0 \end{cases} \tag{17}$$

After some number of iterations we have to decrease the temperature of the simulation. Stability criteria for decreasing the temperature have to be defined.

7 Heuristic Improvement

We have already proposed ideas that can help us to solve a subway design problem. In this section the ideas are combined to arrive at algorithms which

can solve our problem. Some new ideas to improve our algorithms are also presented here.

7.1 Initial Solution

Although the convergence proof of the local search algorithms does not depend on the selection of the initial state, the time cost of the algorithm can be dramatically decreased by selecting a good solution at the beginning. It is obvious that if we start our program with the best solution — optimal subway design, the computational time is reduced to the minimum. We may also apply some cheap heuristics that can approximate our solution very quickly in order to start our algorithm with initial state as near as possible to the best solution. We have formulated four cheap heuristics which are reasonable.

1. **Manually created.** It seems to be a good idea, to create free-hand reasonable solutions by experts.

2. **Local minima from HC.** The Hill Climbing is a fast algorithm in finding local minima. Especially Simulated Annealing algorithms can be improved by selecting these local minima as the initial state because they are very expensive.

3. **Best 2 and 3-stop subways.** Selecting some N best subways of the 2-stop subways (Central Station and secondary stop) and 3-stop subways (Central Station and two additional stops) is really very cheap and seems to be a good first approximation of the future subway localization.

4. **Shortest path pre-selection.** Choose the subway s that minimizes $\sum_{a,b \in E_s}(\omega_{ab} - T(G_0))$ and satisfies the given conditions. You can formulate this as a LP-problem and it can be solved in reasonable time. The criteria function gives you a rough estimation whether connections are important or not.

7.2 Neighbourhood Definition

The definition of the neighbourhood relation is a very important requirement for constructing a correct local search solution for the problem. If the relation of the neighbourhood is introduced in accordance with variability of subway properties, we can expect that local search algorithms will be able to detect an optimal solution. Relation of the neighbourhood should at least guarantee

cohesion of the solution space, however this relation is important for the correct convergence of the local search process to the optimal solution. We are presenting two possible definitions of the neighbourhood structure:

1. Adding and deleting nodes,
2. Exchanging nodes.

7.2.1 Adding and Deleting Nodes

Let the subway line be represented by a sequence of stops. The subway lines $s_1 = (n_1, \ldots, n_k)$ and $s_2 = (n_1, \ldots, n_{j-1}, n, n_j, \ldots, n_k)$ are neighbours. We can see, that subway s_1 is a modified version of subway s_2 by deleting stop n, and also s_2 is a modified version of subway s_1 by adding stop n between n_{j-1} and n_j. In other words, we can say that we have two operations of the local change in our neighbourhood structure. With this structure it is possible to go from one solution to another in a finite number of steps.

7.2.2 Exchanging Nodes

Let the subway line be represented by a sequence of stops. We have two subway lines $s_1 = (n_1, \ldots, n_k)$ and $s_2 = (n_1, \ldots, n'_j, n_{j+1}, \ldots, n_k)$. s_1 and s_2 are neighbours when $n_j \neq n'_j$. We can see that subway s_1 is subway s_2 with the j-th stop is changed. In other words we have one operator that describes the local change. We can get each stop from the current subway design and exchange it with another, that currently does not belong to the subway design. With this structure you can get every subway with a fixed length.

7.3 Reducing of the Neighbourhood

The size of the neighbourhood is of $\mathcal{O}(Nk)$. Because the evaluation of the criteria function $T(G)$ is very expensive it is recommendable to reduce the size of the neighbourhood. If you do a smart pre-selection, the complexity of your algorithm decreases without loss of efficiency. So it will faster converge to a good approximation.

Assume that we want to add a subway stop b between stop a and c. As we remember, in Section 5, we have defined an equation with the minimum that can evaluate measures for the subway with stops abc from measures of subways with stops ab, bc and ac.

$$t_{ij}^{(abc)} = \min(t_{ij}^{(ab)}, t_{ij}^{(bc)}, t_{ij}^{(ac)} + t_{ab} + t_{bc} - t_{ac}) \tag{18}$$

Since

$$\min(x, y, z) \leq \frac{x + y + z}{3} \tag{19}$$

holds true for all $x, y, z \in \mathbb{R}$, we obtain

$$t_{ij}^{(abc)} \leq \frac{t_{ij}^{(ab)} + t_{ij}^{(bc)} + t_{ij}^{(ac)} + (t_{ab} + t_{bc} - t_{ac})}{3}. \tag{20}$$

Because addition can be get out of the sum, we can write that

$$\omega_{abc} \leq \frac{\omega_{ab} + \omega_{bc} + \omega_{ac} + C(t_{ab} + t_{bc} - t_{ac})}{3} = u_{abc}. \tag{21}$$

Where $C = \sum c_{ij}$, can be pre-calculated before. So, now we have a formula, that can be evaluated more quickly than real subway measures. With the evaluation of the value u we can avoid $3 \cdot N^2$ multiplications, which is very attractive. We can use the value u as a pre-selector, that does some sort of ranking of the possible local changes.

The application of this ranking function is clear for HC and TS algorithms, we can cut-off the neighbourhood from 600 to 20-200 best candidates. The situation is not so clear for the SA algorithm. Firstly, we can do SA without any modifications, because SA itself selects only one local change. Secondly, we can use u for increasing probability of the selection of better candidates in a proportional way (roulette selection). However we believe that it can destroy the property of SA of avoidance of local optima.

8 Conclusions and Recommendations

8.1 Conclusions

- We have made a mathematical model that describes the problem of finding the subway that minimizes a balance between costs and travel times.
- An approach that can find a good solution for this problem, where the costs and buslines are fixed, are designed.

8.2 Recommendations

- The approach should be generalized. Such that it is possible to change the buslines. Another generalization step could be to make it possible to change the number of busses for a busline.

- This generalized approach must be implemented into a user friendly program. The user should have the possibility to make some choices manually.
- In order to decide how to design the subway line a couple of examples should be tried without exceeding the available amount of money for the costs. In this way a network will be designed that gives a good balance between costs and travel time.

References

[1] E. Minieka, "Industrial engineering, Optimization Algorithms for Networks and Graphs".

[2] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs" Springer Verlag, Berlin, 1996.