J Heuristics (2014) 20:561–587
DOI 10.1007/s10732-014-9252-3

# Mind the gap: a heuristic study of subway tours

**M. Drozdowski · D. Kowalski · J. Mizgajski · D. Mokwa · G. Pawlak**

Received: 13 June 2013 / Revised: 19 March 2014 / Accepted: 18 June 2014 /
Published online: 9 July 2014


**Abstract** What is the minimum tour visiting all lines in a subway network? In this paper we study the problem of constructing the shortest tour visiting all lines of a city railway system. This combinatorial optimization problem has links with the classic graph circuit problems and operations research. A broad set of fast algorithms is proposed and evaluated on simulated networks and example cities of the world. We analyze the trade-off between algorithm runtime and solution quality. Time evolution of the trade-off is also captured. Then, the algorithms are tested on a range of instances with diverse features. On the basis of the algorithm performance, measured with various quality indicators, we draw conclusions on the nature of the above combinatorial problem and the tacit assumptions made while designing the algorithms.

**Keywords** Graph cycles · Urban railway · One-pass heuristics · Metaheuristics · Evaluation of heuristics

## 1 Introduction

Consider a set of lines embedded in a ground communication network. What is the shortest cycle visiting all the embedded lines? For example, given the Tube railway network, with all its different color lines, is it possible to visit all the lines in a day? The above problem originally arose as a touristic challenge (Drozdowski et al. 2012). More generally, the ground communication network can be a railway, bus or airline network. Various institutions or companies use the ground networks to establish their own communication sub-networks as logistic networks, public transportation lines, etc. A sub-network is visited by traversing at least one of its arcs. The question is,

M. Drozdowski (✉) · D. Kowalski · J. Mizgajski · D. Mokwa · G. Pawlak
Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland
e-mail: Maciej.Drozdowski@cs.put.poznan.pl

what is the shortest cycle visiting all the embedded sub-networks to deliver some piece of information, a virus, or a cure, to all the sub-networks. Our problem models transfer processes which happen in passage. For example, spread of diseases and rumors between people, exchange of viruses or advertisements between mobile devices on the move while sharing means of transport. By the fact that not one person, but streams of people are traversing the arcs, and trains are changed at the adjacent nodes, the lines/colors are visited, in a sense, in parallel. Thus, cycles indicate a reservoir spreading information, amplifying disease in the transportation systems. A short cycle means lower costs in advertising or fast spread of infections in disease dynamics modeling. Yet another example is the shortest circuit for an inspector to visit all the sub-networks. For simplicity of the further exposition we will use the metaphor of subway, such as Tube, and will call this problem *Mind The Gap* (MTG for short).

Combinatorial optimization and operations research have a rich heritage of problems related to graph circuits. Thus, it may seem that MTG could have been studied in some form already. Certainly, MTG is not equivalent to the Traveling Salesman Problem because we do not have to visit each node once, but each line (sub-network) at least once. In covering salesman problem (Arkin and Hassin 1994; Current and Schilling 1989) a salesman is to visit a subset of the cities but it suffices to enter into some neighborhood of the city. MTG is not a Covering Salesman Problem because we visit lines, not the cities. Furthermore, covering the neighborhood introduces the idea of locality which is not present in MTG, because subway lines can stretch across whole cities. In group-TSP (Safra and Schwartz 2006) the salesman should visit at least one node from each given set. Again, it is not MTG because we have to visit representative members of sets of arcs, and the sets must be connected. Orienteering problem (Vansteenwegen et al. 2011) arose in the context of the orienteering game. In this game a set of control points (the nodes) is given, each with a certain reward. Traveling between the control points takes time represented by arc weights. A player starting in a specified control point must return within a specified time maximizing the rewards collected in the visited control points. Still, in MTG we have to visit arcs, not the nodes. Spatial distribution of the reward in MTG is different because in the orienteering problem the reward is bound to the nodes, in MTG it is associated with connected sets of arcs. Moreover, in the Orienteering Problem the reward is a number and there is a trade-off between the gain and circuit length, while in MTG each line has to be visited. In traveling purchaser problem (Laporte et al. 2003) a set of products is available in each city. The products in a city are available in limited quantity at predetermined costs. The purchaser has to buy a given number of units of the products from a given list. A minimum-cost route must be constructed where the cost is a sum of the travel time, and the purchase costs. There are more intricate differences between the two problems than just the fact that in traveling purchaser problem we collect products in the cities, and in MTG line colors in sub-networks. Namely, the distribution of the products in the cities can be arbitrary in the traveling purchaser problem, while in MTG the ”product” is a line with a specific topology. Furthermore the cost and quantity of the product in the purchaser problem can be arbitrary. In MTG cost of a ”product” in an arc is zero, and quantity is binary. Euler cycle (Eulero 1741) and Chinese Postman problems (Black 2012; Garey and Johnson 1979) consist in visiting each edge once or at least once, respectively. However, in our problem we do

not have to visit all edges. Just a subset of edges covering all the different color lines suffices. In the rural postman problem (Eiselt et al. 1995) a set of arcs in a graph is to be traversed in a shortest cycle. MTG is not a Rural Postman Problem because instead of a particular arc, a set of alternative arcs (the lines) can be visited. What is more, the arcs to be visited by the rural postman can be arbitrarily scattered in the network, and disconnected. In MTG all arcs belong to some line(s), and lines are connected sets of arcs. Hence, despite superficial similarities MTG is qualitatively different from the above classic problems.

*Previous Work* MTG problem has been studied in Drozdowski et al. (2012). It has been observed that depending on the level of resolution, the structure of real subway lines can be quite diversified. They can be trees, cycles, there can be parallel connections between two stations. Some stations may be accessible only in one direction. In general a line can be considered a set of directed connected cycles. MTG has several versions: with arbitrary or with equal arc weights, symmetric or asymmetric, with a given starting node or without. In the last case the *universally* shortest circuit, i.e. a circuit starting in any node, is searched for. Symmetric MTG with equal arc weights is **NP**-hard even if the lines are trees. With arbitrary weights symmetric MTG is **NP**-hard if the lines are simple paths. Two branch and bound algorithms, an algorithm based on an ILP model, greedy heuristics CUL, CMEE (see Sect. 3) were proposed and evaluated on the instances of 12 cities. The algorithm using an ILP model and CPLEX turned out to be time-inefficient, and was outperformed by the branch and bound algorithms. For CMEE and CUL heuristics lower bounds on the worst case performance ratio were provided. These heuristics were fast, but their solutions were far from the optima. Branch and bound algorithms provided the best solutions, but their runtimes were barely acceptable. Hence, alternative algorithms are necessary to solve bigger instances and to conduct a study of the MTG problem features.

The goals of the study in this paper are as follows: (i) propose a set of heuristics which are fast and build good solutions for MTG problem, (ii) evaluate how the heuristics strike a balance between quality and cost, (iii) determine what features of MTG instances induce hardness or easiness in heuristic solving. As byproducts (iv) a technique of evaluating the trade-off of quality versus runtime and (v) a method of searching for difficult instance features emerged. In this study we make a tour d'horizon of heuristic methods solving a new combinatorial optimization problem. We propose the way of reasoning about MTG solutions and compare a broad set of algorithms. Consequently, we will not elaborate in-depth on fine-tuning specific metaheuristics or their alternative implementations. A comprehensive evaluation of sensitivities and options for alternative implementations must be deferred to further research. We study here the universal MTG problem in symmetric weighted networks. Test instances are the subway networks of selected cities and a set of simulated networks. Further organization of the paper is as follows. In Sect. 2 MTG problem is formulated. Section 3 is dedicated to low-order complexity algorithms. Local search methods and metaheuristics are introduced in Sect. 4. A network simulator constructing test instances is described in Sect. 5. The results of algorithm evaluation are presented and discussed in Sect. 6. The notation used throughout the paper is summarized in Table 1.

**Table 1** Summary of notation

| | |
|---|---|
| $A$ | Set of arcs (interstation connections) |
| $A(i)$ | Set of arcs constituting line $i$ |
| $ad(a, b)$ | Distance between arc $a$ and arc $b$ (defined in the text) |
| $d(a)$ | Length of arc $a$ |
| $sd(u, v)$ | Length of the shortest path from node $u$ to node $v$ |
| $G(V, A)$ | Subway network |
| $(i, j)$ | Arc from node $i$ to node $j$ |
| $l(X)$ | Set of lines (different colors) visited in set $X$ of arcs |
| $L$ | The set of lines (different colors) |
| $lra(i)$ | Line $i$ representative arc |
| $m$ | Number of arcs |
| $n$ | Number of nodes (stations) |
| $V$ | Set of nodes (stations) |

## 2 Problem formulation

The subway network (the ground transportation network) is a directed graph $G(V, A)$, where $V$ is the set of $n$ nodes representing stations and $A$ is the set of $m$ arcs representing interstation connections. For arc $a \in A$ weight $d(a)$ corresponds to its travel time. Each arc belongs to some line. The set of lines traversing arc $a$ will be denoted $l(a)$. More generally, $l(X)$ will denote the set of lines in set $X$ of arcs, i.e. $l(X) = \bigcup_{a \in X} l(a)$. $L$ is the set of lines (different colors) embedded in the subway network. Line $i = 1, \ldots, |L|$ is defined by the set of arcs $A(i)$ covered by the line. The arcs in $A(i)$ are connected. It means that nodes incident with arcs in $A(i)$ are connected by the arcs in $A(i)$. Line $i$ is considered visited by the tour if the tour traverses at least one arc in $A(i)$. It is required to construct the shortest circuit visiting all lines. Since we allow to start the circuit from an arbitrary node, we call it universal MTG circuit (uMTG). In this paper we consider symmetric MTG. It means that $\forall (i, j) \in A : d(i, j) = d(j, i), l(i, j) = l(j, i)$. Therefore, it may be convenient to refer to pairs of arcs $\{(i, j), (j, i)\}$ as to edges. Still, where needed, we will refer to arcs for precision.

A solution of an MTG problem is a cycle of arcs where certain lines are visited. To build a solution, it is necessary to choose for each line an arc on which the line is visited. For line $i$ we will call such an arc *line $i$ representative arc* ($lra(i)$). Obviously, if $a = lra(i)$ then $i \in l(a)$. Since $l(a)$ may comprise many lines, arc $a$ may represent many lines. A solution may be perceived as a sequence of different line representative arcs connected by the shortest paths, rather than as a sequence of nodes. Arc-to-arc distance from arc $a = (w, x)$ to arc $b = (y, z)$ is defined as $ad(a, b) = sd(x, y) + d(y, z)$, where $sd(x, y)$ is the shortest path distance from node $x$ to node $y$. Note that in general the distance between two arcs is asymmetric, i.e. in general $sd(x, y) + d(y, z) \neq sd(z, w) + d(w, x)$.

The representation of a solution as a sequence of line representative arcs connected by the shortest paths will be called *arc-vertex representation*. Arc-vertex representation $H(V', A', L)$ of the input instance is constructed as follows:

- An arc $(x, y) \in A$ becomes a vertex $v_{xy} \in V'$.
- The set of lines $l(x, y)$ becomes a set of lines $l(v_{xy})$ associated with vertex $v_{xy} \in V'$.
- For each pair $(w, x), (y, z) \in A$ an arc $(v_{wx}, v_{yz}) \in A'$ of length $sd(x, y)+d(y, z)$ is introduced.

Arc-vertex representation may be ambiguous because many shortest paths may exist between pairs of vertices of the original subway graph. To avoid uncertainty in the solutions some algorithms ignore colors visited on the shortest paths connecting the line representative arcs. Then, the set of lines covered by a set of arc-vertices is independent of their permutation. For example, $l(v_{ab}, v_{de}, v_{ed}, v_{fg}) = l(v_{de}, v_{ab}, v_{fg}, v_{ed})$. It simplifies preserving feasibility of the solutions, particularly in local search algorithms. Still, some other algorithms take advantage of visiting additional colors on such shortest paths. We outline it in Sects. 3 and 4.

Let us observe that in symmetric MTG, long chains of edges can be contracted to one edge to simplify the solution process. We will call such instances *shortened*. The following conditions must be met to contract a node with the two incident edges to one edge (Drozdowski et al. 2012): (i) the station is connected with exactly two other stations, (ii) the two neighboring stations must be on the same lines, (iii) the station must be more than one edge away from any crossing with other line.

## 3 One-pass algorithms

The methods introduced in this section are greedy one-pass algorithms with well-defined complexity. Two algorithms add new lines to existing circuits, the third algorithm merges cycles, the fourth builds a path and eventually closes it.

### 3.1 Furthest line first (FLF)

In this algorithm new line representative arcs are inserted between pairs of the already visited lines. The lines to be inserted are considered in the order of decreasing distance from the circuit. A distance between two lines $j, k$ is the shortest distance between any pair of arcs belonging to $j$, and $k$. If lines $j, k$, use the same arc $(u, v)$, i.e. $\{j, k\} \subseteq l(u, v)$, then the distance between the two lines is $d(u, v) + d(v, u)$ because the lines are visited by traversing the arc. FLF starts with an initial cycle of two furthest lines $j, k \in L$. It means that the two closest arcs $(u, v) \in A(j)$, $(w, x) \in A(k)$ are embraced in a cycle of length $d(u, v) + sd(v, w) + d(w, x) + sd(x, u)$.

Suppose a cycle joining $i \geq 2$ lines is built. The partial cycle is a sequence of $i$ line representative arcs connected by the shortest paths. For the $|L| - i$ lines (colors) remaining out of the cycle, the shortest distance to the cycle is calculated. In more detail, for some line $j$ not yet in the cycle insertion of each arc $a \in A(j)$ between each pair of the $i$ arcs representing the partial cycle is verified. As a result, arc $a' \in A(j)$ the closest to the partial cycle becomes a potential $lra(j)$. It is also known where $a'$ should be inserted in the partial cycle. Line $j'$ with the biggest distance to the partial cycle is selected, and arc $lra(j')$ is inserted in the appropriate position in the sequence

of $i$ arcs. Thus, a cycle of length $i+1$ is built. Additional lines visited by chance, either on arc $lra(j')$ or on the shortest paths connecting $lra(j')$, are included in the cycle and are not considered in the later steps for re-insertion. Consequently, in the $i$th iteration the cycle may comprise more than $i$ different colors. This procedure is continued until including all lines in the solution. The complexity of this method is $O(n^3 + |L|^2 m)$, where $O(n^3)$ results from calculating distances between the arcs, the cycle is expanded at most $|L|$ times requiring verification of insertions between at most $|L|$ lines in the cycle, while each insertion involves checking at most $O(m)$ candidate arcs.

### 3.2 Cycle development (CD)

Cycle development algorithm (CD) relies on a similar principle as FLF. It augments the partial cycle by inserting an unvisited line between the most convenient pair of lines already in the circuit. Let $C$ be the current partial circuit. Then, $l(C)$ denotes the set of colors visited in $C$ and $l(a)$ is the set of colors in some arc $a$. Algorithm CD chooses for insertion such arc $a \notin C$ and a pair of neighboring arc-vertices $s, t \in C$ which maximize $|l(a) \setminus l(C)|/(ad(s,a) + ad(a,t) - ad(s,t))$. Thus, CD heuristic chooses the arc and the position that maximize the ratio of the number of unique lines added to the solution to the insertion cost. Unlike FLF, this algorithm ignores any new lines visited on the shortest paths connecting the line representative arcs. The construction of the circuit is initiated from every arc, and the best cycle is chosen. The complexity of CD is $O(n^3 + |L|^2 m^2)$ because it is started from each of the $m$ arcs. Algorithm CD may be started from some given set of arcs if used as a component of other algorithms (cf. Sect. 4). In such cases CD is run once without restarting.

### 3.3 Merge most colorful (MERC)

Algorithm MERge most Colorful expands the circuit by joining pairs of partial cycles covering the greatest number of colors, and in the case of a tie, the closest pair. A different version of this heuristic called MERS (MERge Shortest) joining the closest cycles has been designed. However, it was outperformed by MERC. For conciseness we present MERC method only.

The initial set of cycles is constructed by joining all pairs of lines. Line representative arcs are connected by the shortest paths. Additional colors visited on the shortest paths joining line representatives are marked as visited and no longer considered available for joining. Thus, in the initial and in the following steps the number of colors comprised in a cycle is not necessarily a power of two. Next, pairs of cycles are chosen in the order of decreasing number of comprised colors. If there is a tie, then a shorter cycle is selected. Cycle merging is repeated until including each line in some cycle. If a line remains without a complement to build a pair, then the line forms a cycle only with itself represented by its shortest arc. In the following steps, the procedure is repeated: All pairs of cycles are merged. The most colorful cycle pairs are selected until covering all colors. The algorithm stops when only one cycle remains.

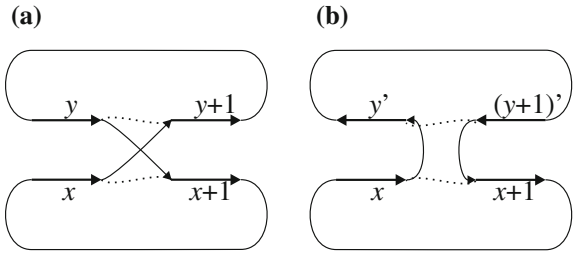


**Fig. 1** Merging algorithms **a** Merging without changing direction of a cycle. **b** Merging with inverting a cycle. $y'$, $(y+1)'$ are inverted arcs $y$, $y+1$

Partial cycles are merged by the following procedure. Suppose $x, x+1$ are line representative arcs of two lines visited consecutively in one cycle, and $y, y+1$ are two line representative arcs of two lines visited consecutively in the other cycle. The algorithm tries two ways of joining the cycles. First, by traversing from $x$ to $y+1$, and from $y$ to $x+1$ such that the sum of lengths of the two cycles increases by $ad(x, y+1) + ad(y, x+1) - ad(x, x+1) - ad(y, y+1)$ (cf. Fig. 1a). The second joining method attempts inverting direction of one cycle, which is admissible in the symmetric MTG. Let $y'$, $(y+1)'$ be the inverted arcs $y$, $y+1$. Arc $x$ is connected with $y'$, and $(y+1)'$ with $x+1$ (cf. Fig. 1b). Then the overall length of the solution increases by $ad(x, y') + ad((y+1)', x+1) - ad(x, x+1) - ad(y, y+1)$. This procedure is repeated for all line representative arcs $x$, $y$ in the considered cycles. The complexity of this method is $O(n^3 + |L|^2 m^2 + |L|^4)$ because calculating distances between arcs requires time $O(n^3)$, the initial set of cycles requires choosing from at most $O(|L|^2)$ pairs of lines represented by at most $O(m^2)$ pairs of arcs. In the worst case $O(|L|)$ merging iterations are needed to include all colors in a single cycle, where each iteration involves choosing at most $O(|L|)$ partial cycles from a set of $O(|L|^2)$ cycle pairs.

### 3.4 Closest most efficient edge (CMEE)

Closest most efficient edge (CMEE) introduced in Drozdowski et al. (2012) extends a partial solution, which is a path, by appending an arc with the greatest ratio of the number of new colors to the distance needed to traverse the arc. Suppose $v$ is the last vertex of the subway graph in the current partial solution $C$. Arc $(x, y)$ maximizing $|l(x, y) \setminus l(C)|/(sd(v, x) + d(x, y)))$ is appended to $C$. Ties are broken arbitrarily. When all lines are visited the circuit is closed by the shortest path returning to the starting node. CMEE is started from all vertices of the initial subway graph, and the best solution is chosen. Complexity of this method is $O(n^3 + |L|mn)$.

### 3.5 Random (RND)

Algorithm random builds a random sequence of colors. For each color $i$ a random arc from $A(i)$ is chosen as $lra(i)$. Line representative arcs are connected using the shortest paths. This method is used as a reference to verify whether other algorithms build useful solutions. Were the quality and runtime of random algorithm comparable with other algorithms, one could conclude that such algorithms are useless. Complexity of this algorithm is $O(n^3 + |L|)$.

## 4 Local search-based methods

In this section we propose various algorithms based on local search. We start with description of neighborhood types. The neighborhoods are defined in the arc-vertex representation. Then we proceed to the algorithms. For simplicity of presentation we assume that $|L| < m$.

### 4.1 Neighborhoods

#### *4.1.1 Vertex reversal*

In the symmetric MTG, $\forall(i, j) \in A : l(i, j) = l(j, i)$ and in the arc-vertex representation $l(v_{ij}) = l(v_{ji})$. Therefore, arc-vertices $v_{ij}, v_{ji}$ can be exchanged without affecting the feasibility of the solution. Still, such reversal may change quality of the solution because the circuit arrives at the edge in a different node of a subway graph. The size and complexity of evaluating this neighborhood is $O(|L|)$, i.e. the number of line representative arcs in the solution.

#### *4.1.2 Vertex removal*

Some solutions contain lines covered many times. Arc-vertices comprising only lines present in other arcs are removed from the solution. When choosing an arc to remove, the arc which decreases the length of the solution the most, is preferred. In case of a draw we remove a less colorful arc. The size of this neighborhood is $O(|L|)$ and the complexity of evaluating it is $O(|L|^2)$.

#### *4.1.3 Vertex exchange*

Vertex exchange attempts to replace an arc-vertex with a better one. For each arc-vertex $x$ a list $sub(x)$ of substitute arcs is prepared. List $sub(x)$ includes arcs comprising all the lines uniquely covered in $l(x)$. Let $x-1, x+1$ be the two arc-vertices, respectively, preceding and succeeding $x$ in the current solution (cf. Fig. 2). Let $y, y+1$ be two arc-vertices visited consecutively in the current solution and different from $x$. If arc $a \in sub(x)$ is inserted between $y, y+1$ and $x$ is removed from the solution, the circuit length increases by $ad(x-1, x+1) + ad(y, a) + ad(a, y+1) - ad(x-1, x) - ad(x, x+1) - ad(y, y+1)$. Arc $a \in sub(x)$ and insertion position $y$ with the minimum circuit length are selected. The move is considered feasible if the circuit length is reduced. The size and cost of evaluating this neighborhood are $O(|L|^2 m)$ because $|L|$ line representative arcs may be replaced with $O(m)$ new arcs, considered for insertion in $O(|L|)$ new positions.

#### *4.1.4 Vertex swap*

Vertex Swap alters the permutation of line representative arcs in the solution by performing pairwise swaps. For example, sequence $\ldots, x-1, x, x+1, \ldots, y-1, y, y+$

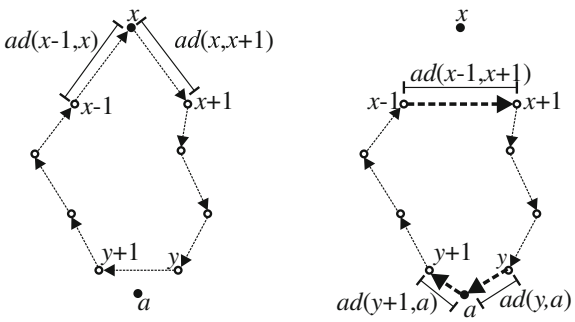


**Fig. 2** Vertex exchange of arc-vertices $a$ and $x$

$1, \ldots$ of line representative arcs can be replaced with $\ldots, x-1, y, x+1, \ldots, y-1, x, y+1, \ldots$. The size of the neighborhood is $O(|L|^2)$.

#### *4.1.5 Sub-path interchange*

This neighborhood structure is inspired by the 2-Opt method (Croes 1958; Lin and Kernighan 1973) for the Traveling Salesman Problem. Let $(x, x+1)$ and $(y, y+1)$ be two pairs of different line representative arcs visited consecutively in the circuit. They separate sub-paths $(x+1, \ldots, y), (y+1, \ldots, x)$. Since we consider symmetric MTG, inversion of the sub-path $(x+1, \ldots, y)$ is possible. It is checked whether traversal $x, y, \ldots, x+1, y+1, \ldots, x$ results in a shorter cycle. All possible pairs of line representative arcs $x, y$ are verified. The pair $x, y$ reducing circuit length the most is applied to obtain a new circuit. The size of the neighborhood is $O(|L|^2)$.
Algorithms using the above neighborhoods are presented below.

### 4.2 2-Opt

This algorithm is a 2-Opt method (Croes 1958; Lin and Kernighan 1973) operating on arc-vertices rather than on nodes of the subway graph. We start with a solution built by MERS algorithm. Then, Sub-Path Interchange move is repeated as long as there is any improvement in the circuit length. 2-Opt is an example of a simple local search-based algorithm, using just one type of the neighborhood improving interconnection of line representative arcs.

### 4.3 Local search (LS)

Algorithm local search starts from a solution constructed by the CD algorithm (Sect. 3.2). Given the current solution, all moves in all neighborhoods are evaluated and the move yielding the greatest quality improvement is applied. The process is repeated until no improving move can be found.

### 4.4 Multistart local search (MSLS)

Multistart Local Search algorithm is an extension of Local Search based on restarting the search from scratch upon finding a local optimum. As a method supplying starting solutions we used randomized algorithm CD. In more detail, a random permutation of $m$ arcs is generated. From this permutation $k$ initial arcs are used as a starting partial circuit for algorithm CD. The number of initial arcs $k$ is iteratively increased from 1 until covering all $|L|$ lines by the initial set of the $k$ arcs. Then a new permutation of $m$ arcs is generated and the procedure is repeated. MSLS stops when time limit is reached.

### 4.5 Iterated local search (ILS)

Iterated local search method is based on the assumption that local optima are often clustered (Martin et al. 1992; Codenotti et al. 1996). Rather than restarting the search from an entirely new solution, as in MSLS, the local optimum is perturbed, hoping that the perturbed solution $s'$ will lead to a different local optimum. Then LS algorithm is restarted from $s'$.

In our implementation the perturbation intensity is controlled by the fraction $\kappa$ of line representative arcs removed from the solution, i.e. $\kappa x$ arc-vertices are extracted from the solution comprising $x$ arc-vertices. If consecutive iterations produce the same solution, then $\kappa$ is increased. Otherwise, it is decreased to exploit the chance of finding better solutions similar to the current one. To preserve potentially good parts of the solution, $x\kappa$ *consecutive* line representative arcs are removed starting from some randomly chosen arc-vertex $t$. In order to restore the feasibility of the solution, line representative arcs are drawn randomly for each missing line and are inserted consecutively after arc-vertex $t$. Thus, a new sequence of arc-vertices is introduced in the position of the removed sequence. ILS starts from a solution generated by 1) constructing an initial seed circuit of three randomly chosen arcs, 2) completed to a full cycle by algorithm CD, and 3) optimized by algorithm LS.

### 4.6 Hybrid genetic algorithm (HGA)

Hybrid Genetic Algorithms combine advantages of local search methods and evolutionary schemes provided by genetic algorithms (Talbi 2002; Abdullah et al. 2012; Berlińska et al. 2009). Since this is the first attempt to solve MTG problem and there are no recommendations on HGA implementation, we chose simple evolutionary scheme and operators. A Lamarckian model of evolution was used, i.e. solutions obtained by applying the genetic operators were subjected to local optimization prior to evaluation. Any subsequent operations were performed on the optimized solutions.

A solution (chromosome) in our HGA is a sequence of arc-vertices. A recombination operator transforms two parent solutions with the aim of preserving parts of the arc-vertex sequences appearing in both parents (see Fig. 3). Thus, arc-vertex subsequences shared by both parents are identified first. The sub-sequences are passed to two children solutions. One child inherits the permutation of the sub-sequences

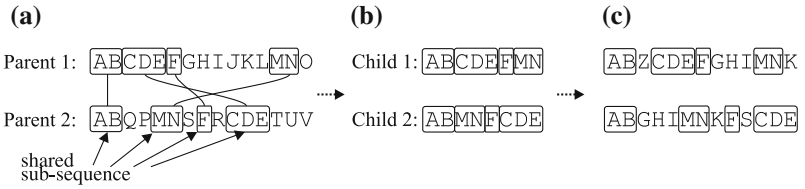


**Fig. 3** HGA recombination operator. **a** Parent solutions, **b** shared sequence extraction, **c** reconstruction of feasibility by CD

from the first parent and the other child inherits the sub-sequence permutation from the second parent. The shared sub-sequences may cover fewer than $|L|$ lines. Then, feasibility of the offspring is restored by CD algorithm starting from the shared sub-sequences as initial partial circuits. However, the algorithm is not allowed to insert a new arc-vertex into the shared sub-sequence.

The solution perturbation routine introduced in ILS algorithm (Sect. 4.5) is our mutation operator. The perturbation intensity $\kappa$ is fixed at $\kappa \approx 0.18$. Intensity of mutation is controlled by probability of mutation $\mu$. Mutation probability $\mu$ is increased in discrete steps whenever the current best solution is not improved in the subsequent generations. Similarly, it is decreased when a better solution is constructed.

A population of 50 solutions is maintained. In order to apply recombination, pairs of solutions are drawn at random from the population, without returning to the solution pool, until all solutions have been drawn. All solutions constructed in the recombination stage enter the mutation stage. During the mutation stage all solutions are subject to mutation with probability $\mu$. Elitist selection method is applied. Parents are ranked together with their offspring and the best solutions survive to the next generation.

Members of the initial population are obtained by: (1) generating an initial partial circuit of three randomly selected arc-vertices, (2) augmented by CD algorithm to a circuit comprising all $|L|$ lines, and (3) optimized by LS algorithm.

## 5 Subway network simulator

Testing the above algorithms requires a large number of problem instances. Therefore, a subway network simulator was designed. Before proceeding to the details of the simulator let us outline the rationale behind its construction. Subway graphs cannot be adequately represented by an arbitrary random graph because the real urban railway networks are built with considerable cost and space limitations (Drozdowski et al. 2012). In particular, two nodes cannot be arbitrarily close to each other because stations occupy space. For the same reason node degrees are limited. Subway graphs are not planar, but no arbitrary number of lines can pile one on another. Lines very often share interstation connections. Since lines are built in the real space, travel in no time is impossible, arc lengths must correspond with the length of the rails. Building connections between stations that are spatially very distant is less likely than between two spatially close stations. Changing direction of a line requires space and the lines do not bounce in the city in an arbitrary zigzag. Hence, our subway network simulator emulates the process of building subway lines on the city ground.

The input parameters of the simulator are: the number of lines $|L|$, the number of stations $n$, probability $\alpha$ of using an existing edge, probability $\beta$ of using an existing station. The algorithm simulates building lines one segment (edge) at a time. Lines are simple paths. A line is built along the existing part of the network or new stations and edges are added. Obviously, the first line has only new stations. The new stations arise in a process simulating construction of a route on the plane. Since a line cannot take very tight turns maximal angle between the directions of two segments (edges) of a line is imposed. Building stations very close or very far from each other is not common. Therefore, new stations are created within minimal and maximal distance from the previous station. A new line may follow an existing edge with probability $\alpha$ only if such edge exists. Similarly, an existing station may be visited by the new line with probability $\beta$ only if a station satisfying distance and direction conditions exists.

In the first step stations are randomly distributed to the lines. The stations assigned to a line are created while "building" the line. Suppose we construct line $f$. A random existing station $s$ is selected. Construction of line $f$ starts in station $s$. Thus, each line is connected with the rest of the network. Let $x$ be the number of stations assigned to $f$. A random integer $y$ in range $[1, x]$ is chosen. Station $s$ divides the set of $x$ new stations on line $f$: $y - 1$ new stations are built "before" station $s$, and $x - y$ stations "after" station $s$. If the construction of $f$ arrived at an already existing station, then it can leave the station using one of the existing edges with probability $\alpha$. If the current station is a crossroad, the edge to follow is chosen with equal probability from all possible edges. Were it chosen to detach line $f$ from the existing edges, and stations satisfying the distance and direction limitations exist, then they will be visited with probability $\beta$. All the stations in such a set have equal probability of being picked. Then a new edge is built between the previous and the next already existing stations. If no existing station was selected, then a new station is created at a randomly chosen point within admissible range of distances and directions and line $f$ follows to the new station over a new edge. Observe that if the line arrives at the terminus of the existing network, there are still new stations to be built and there are no stations in the vicinity, then a new station must be created.

This procedure is continued until constructing $x$ new stations. A line may have $x = 0$ new stations which means that the line will be built on the existing stations only. Such a line proceeds in a random walk along the existing edges as long as it is a simple path. In our simulator edge lengths are obtained by calculating Euclidean distances between the stations as float numbers, multiplying them by 1000 and rounding to the closest integer.

## 6 Evaluation of the algorithms

Devising an algorithm for a combinatorial optimization problem is inherently a bicriterial problem of finding a satisfactory trade-off between solution quality and the time cost. Thus, we will analyze algorithm performance in this bicriterial context. Evolution of the solutions in time is an important factor in evaluating performance of the algorithms. On the one hand, the one-pass algorithms introduced in Sect. 3, after certain time, reach the only possible solution for the given instance. On the other hand, meta-

heuristics from Sect. 4 construct many solutions in their runtime. Depending on the time allowance, solution quality may differ. Hence, we have to account for evolution of solution quality in time. Quality of the solutions may be understood in many ways. The average distance from the optimum is a standard quality indicator. However, in a population of instances, distributions of solution quality differ for different algorithms. Thus, an algorithm with good performance on average may fail utterly in certain cases. And vice versa, an algorithm that is weak on average may be the only one to solve certain instances. Hence, other quality criteria are relevant: the number of best solutions found and the distance from the best solution in the worst case. Consequently, we will study not only the average quality, but also the number of best solutions and the worst case behavior.

Unless stated otherwise, test instance parameters were generated as follows: The number of nodes $n$ was drawn from the discrete uniform distribution in range [2, 1000]. The number of lines $|L|$ was drawn from the discrete uniform distribution in range [2, 100]. The probabilities of using the existing edge $\alpha$, and using an existing station $\beta$ were drawn with the uniform distribution from range [0, 1]. In our experiments we evaluated sensitivity of the algorithms to the changes of instance parameters. It was achieved by sweeping through a range of some test parameter (e.g. $n$) and evaluating algorithm performance on populations of instances. For example, for $n = 20$ a population of 100 test instances was generated, each with $n = 20$ and the remaining parameters generated from the above described distributions. Instances for other values of the studied parameter ($n = 2, 5, 10, \ldots$) were generated analogously. Then, it was possible to examine tendencies of the solution quality against the changing test parameter ($n$). We analyzed the sensitivity of the algorithms to:

(A) number of lines $|L| = \{2, 5, 10, 20, 50, 100\}$,
(B) number of nodes $n = \{5, 10, 20, 50, 100, 200, 500, 1000\}$,
(C) probability of using an existing edge $\alpha = \{0, 0.1, 0.25, 0.5, 0.9, 1\}$,
(D) probability of using an existing station $\beta = \{0, 0.1, 0.25, 0.5, 0.9, 1\}$.

Each instance had its shortened version. Thus, 5,200 instances were solved in total. All experiments were performed on a cluster of 30 PCs with Intel Core 2 Quad CPU Q9650 running at 3.00 GHz, with 8 GB of core memory, and OpenSuSE Linux. The algorithms were implemented in Gnu C++. Due to space limitations, only selected results are discussed in the following sections.

### 6.1 Quality versus time trade-off

In this section we analyze how the algorithms perform with respect to the quality versus time cost trade-off. The trade-off is visualized in quality-cost diagram, see e.g. Fig. 4. On the vertical axis relative distance from the best known solution is shown. On the horizontal axis the moment of obtaining a solution is shown. Given a population of instances, a distribution of runtimes and quality indicators was obtained. Hence, in the quality-cost diagrams each algorithm is represented by a box stretching horizontally from the first (Q1) to third quartile (Q3) of time distribution and vertically from the first to third quartile of quality distribution. The point inside a box is a median (Q2)

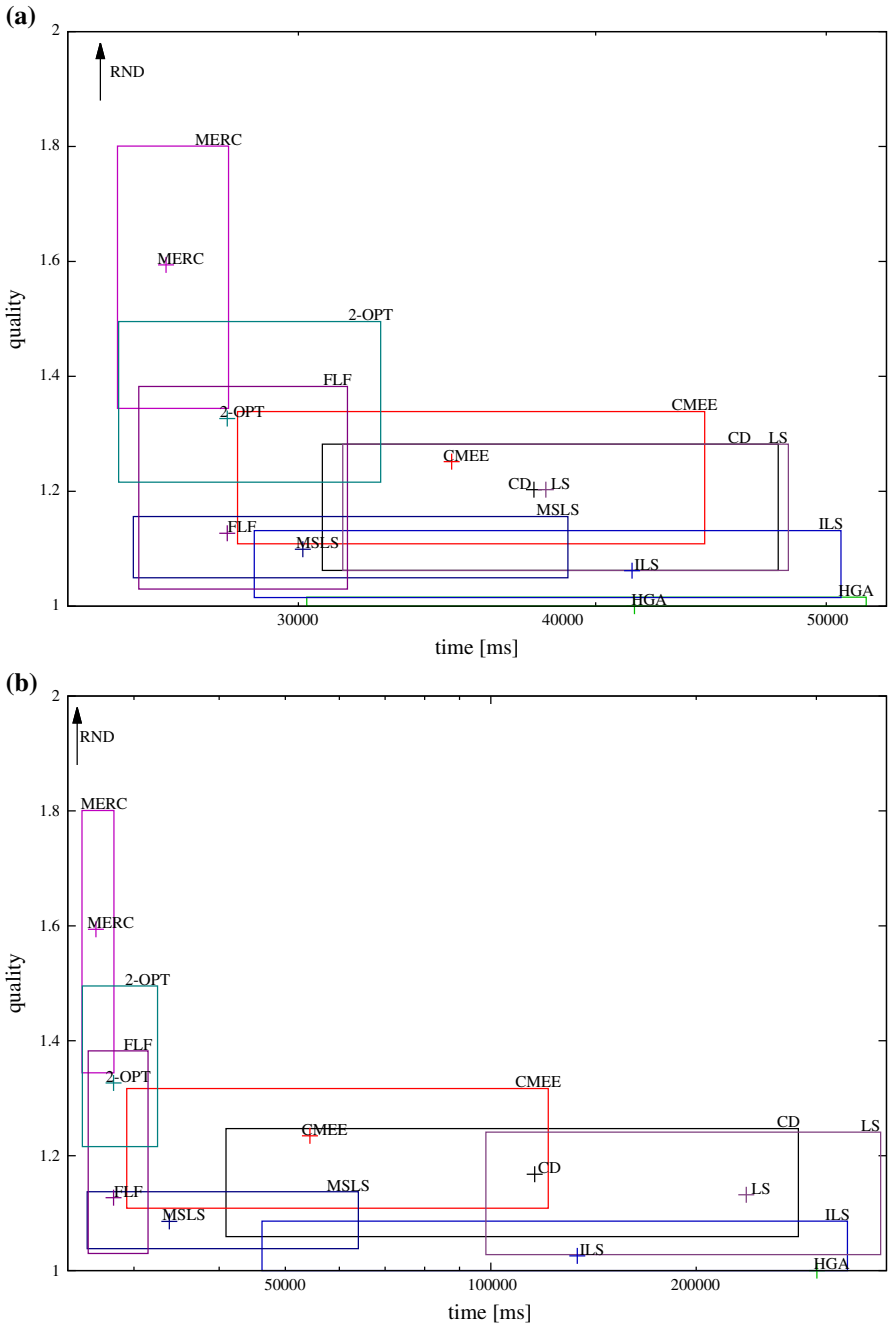


**Fig. 4** Quality versus time cost for $n = 1000$. **a** 60 s time limit, **b** 600 s time limit. (Logarithmic horizontal axis)

of time and quality distributions. In order to capture the time evolution of solution quality, diagrams of quality versus cost are shown for run-time limits: 60 and 600 s.

In Fig. 4 quality-cost trade-off is shown for $n = 1000$ at two time moments: 60 and 600 s. For clarity, position of RND algorithm box is indicated only in Fig. 4 because its time×quality box is [24.5 s, 24.9 s]×[5.99,8.31] both for 60 and 600 s limits. Though it is fast, its quality is substantially inferior to any other algorithm. Conversely, it means that our algorithms do work. In Fig. 4b only median of algorithm HGA is visible because its time×quality box is [62.4 s, 499.1 s]×[1,1]. Thus, algorithm HGA delivers at least 75 % of the best solutions given sufficient time (this will be further discussed in Sect. 6.2). Figure 4 demonstrates that some algorithms trade time for solution quality better than other. It can be verified that algorithms RND, MERC, FLF, MSLS, HGA offer best quality-cost trade-off. Precisely, they are nondominated because there is no other algorithm which improves quality without increasing the runtime. Thus, these algorithms are on the Pareto frontier of quality and cost. Algorithm 2-OPT is also close to the Pareto frontier, but it produces solutions of lower quality than FLF, and has slightly greater runtime dispersion. ILS comes to the Pareto frontier if given enough time (cf. Fig. 4b). On the contrary, algorithms CMEE, CD, LS are dominated. It can be also observed that the dispersion of quality and cost of the nondominated algorithms is related to their position in the Pareto frontier: The fastest algorithms RND, MERC, FLF, 2-OPT have the smallest runtime dispersion, but the biggest quality dispersion. Conversely, algorithm HGA has the smallest dispersion of solution quality, but the biggest dispersion of the runtime. Observe that algorithm FLF has smaller runtime dispersion than MSLS, but bigger quality dispersion. In the further analysis we will concentrate on the location of the quality × time medians (Q2).

Figure 4a, b demonstrate how details of the algorithm design determine the quality-cost trade-off evolution in time. The positions of simple one-pass algorithms like RND, MERC, FLF, 2-OPT remain unchanged because they produced their only solutions within 60 s time frame. Though CMEE and CD are also one-pass algorithms, their quality evolves in time. These two algorithms restart the search many times: CD from each edge, CMEE from each vertex. These processes are time-consuming, and thus, the two algorithms improve their solutions over time.

Algorithms FLF and CD are based on the same idea of inserting new line representative arcs into partially built cycles. FLF takes agile approach because lines visited by chance on the shortest paths joining line representative arcs are included in the solution. CD is unaware of such extra opportunities. Algorithm CD builds $m$ solutions by restarting the construction from each edge as an initial cycle, while FLF builds just one solution starting from the two furthest lines. Comparison of FLF and CD shows that the choice of strategies in FLF is better for quality-cost trade-off.

Quality of solutions delivered by algorithms LS, MSLS, ILS, HGA evolve in time because they are local search algorithms. They use the same neighborhoods, but they differ in choosing the starting solution and the way of diversifying the search. Algorithm LS starts from the solution generated by CD. Since CD takes a lot of time to find its solution, also LS has long runtime. Moreover, in the short run LS is not able to improve solutions of CD (Fig. 4a). With time LS intensifies the search by using all neighborhoods to improve CD's solution, but only with a limited success (roughly 7 % improvement in the median from Fig. 4a to b). Thus, it can be concluded that

CD's solutions are local optima which are hard to escape using our neighborhood structures. This could lead to a conclusion that our neighborhoods are ill-designed. However, other local search algorithms demonstrate that the neighborhoods can be effective if embedded in appropriate control structures. Rather than starting from CD and intensifying the search as LS, algorithms MSLS and ILS diversify the search by restarting it from new solutions (MSLS), or from perturbed local optima (ILS). This approach is better because MSLS and ILS start the search earlier and are able to visit a greater number of solutions. The strategy of restarting the search from new solutions applied in MSLS is more effective in short run because MSLS solutions are delivered faster than ILS's, but even given more time they do not change much (approx. 1.4 % in median from 60 to 600 s). ILS, which perturbs local optima, works better when given more time as it can be seen in both pictures. In a sense, algorithm MSLS learns nothing by restarting the search from scratch, while ILS algorithm preserves parts of the solution which perform like long-term memory. Algorithm ILS is less time consuming than HGA, therefore for longer time frame HGA moves toward later times and ILS is able to come out of being dominated by HGA. Though algorithm 2-OPT is a local search method, its solutions are delivered in 60s time frame and there is no time evolution visible in Fig. 4. 2-OPT starts from a solution delivered by a merging algorithm and improves it by better connecting the line representative arcs. Since the improvement is noticeable (roughly 30 %) it can be concluded that the merging algorithm does not build good line representative arc connections.

Figure 4 shows a typical form of quality vs cost diagram obtained in most of our experiments (A,B,C,D). The overall quality-time relationships between the algorithms changed only in experiment C involving varying probability $\alpha$ of using an existing edge by a new line. In Fig. 5 quality-cost diagram is shown for graphs with probability $\alpha$=1. RND box at [0.6 s,8.7 s] $\times$ [2.77,6.65] is omitted for clarity. The specific case of $\alpha = 1$ means that a new line follows the existing network in a random walk as long as it is possible. Consequently, the required new stations are built mainly from the termini of the existing network. Thus, many lines share edges and the shortest distance between them is zero. It can be expected that algorithms based on line distances (FLF, 2-OPT) may lose their guidance. It can be seen in Fig. 5 that the quality$\times$time boxes of various algorithms to a great extent overlap and the separation of algorithms with respect to quality-cost trade-off is not as clear as before. Thus, the lack of distinction in line distances affects performance of all algorithms. Still, algorithm HGA dominates in quality dimension. Considering medians of quality and time, algorithms MSLS, CMEE, CD, LS, HGA clearly dominate algorithms 2-OPT, FLF, MERC. As could be expected, in the specific type of subway networks generated for $\alpha = 1$, line distances are not a distinguishing feature necessary to guide FLF. Algorithm FLF implicitly assumes that it is inevitable to travel big distances to connect all lines. For similar reasons, algorithm 2-OPT which attempts improving solution on the basis of distance reductions is also not very successful. On the contrary, algorithms like CD, CMEE which assume that good solutions are built on close edges with many colors perform much better. Still, when lines very often share edges and there can be many such edges, the search for the optimum is neither solely guided by the distances, nor solely by the colors. The choice of optimum has more combinatorial, or enumerative, nature. Therefore, CD, CMEE are further outperformed by HGA.

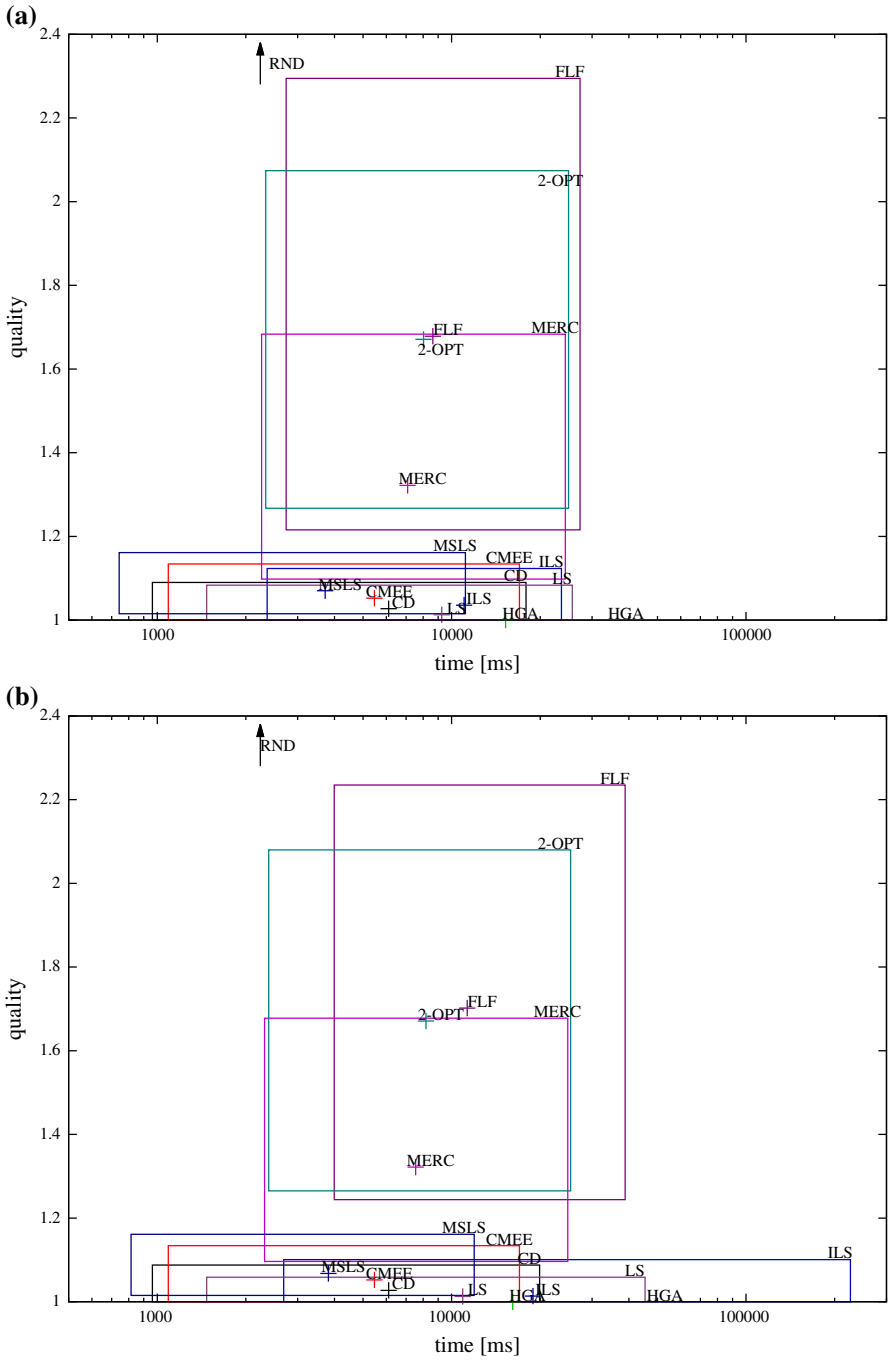


**Fig. 5** Quality versus time cost for $\alpha = 1$ **a** 60 s time limit, **b** 600 s time limit. (Logarithmic horizontal axis)

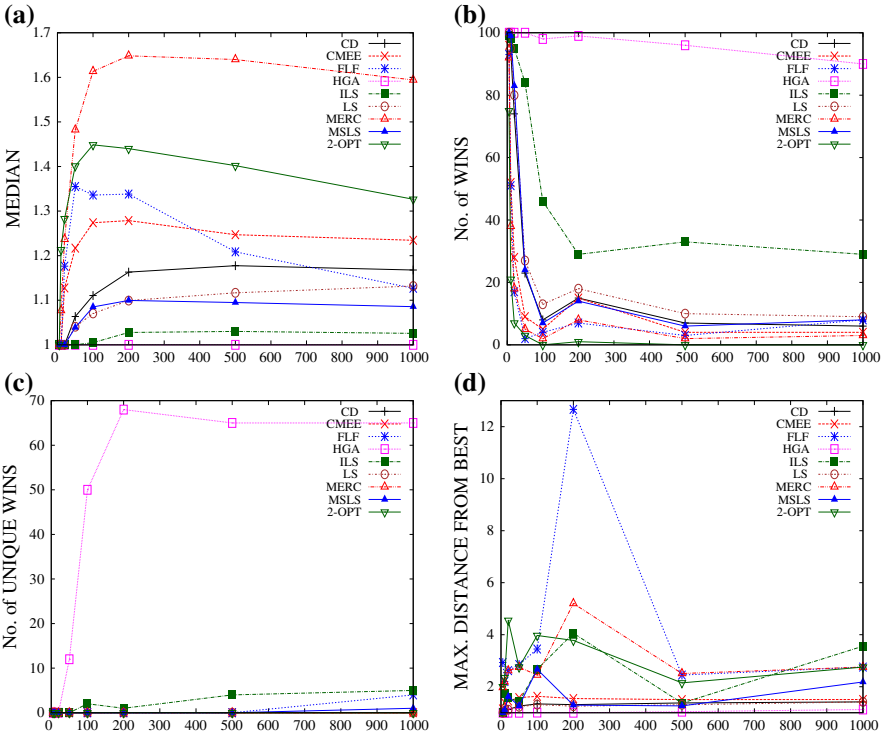


**Fig. 6** Quality versus $n$. **a** Median of the relative distance, **b** number of wins, **c** number of unique wins, **d** relative distance of the worst solutions

## 6.2 Quality versus instance features

In this section we analyze influence of instance features on the quality of the solutions derived by our algorithms. Four quality indicators were used: (a) median of the relative distance from the best solution, (b) the number of the best solutions found (wins), (c) the number of the best solutions found uniquely (unique wins), (d) the worst observed relative distance from the best solution. The unique best solutions were found by one algorithm only, other best solutions could be found by many algorithms. All the algorithms were given 600 s time limit.

In Fig. 6 solution quality is shown against increasing number of nodes $n$ (experiment B). Algorithm RND is not shown because it either fell out of the vertical axis range (Fig. 6a, d), or returned no winning solutions (Fig. 6b, c). It can be seen that algorithm HGA dominates in almost all quality measures. Yet, with growing instance size unique best solutions are gradually found also by other algorithms (Fig. 6c). It could be intuitively expected because HGA is a time-consuming method. Thus, at fixed runtime limit and growing $n$, it can search diminishing solution space. The solutions found by HGA are partially repeated by other methods (Fig. 6b), and ILS is the second best. It is easy to find good solution in small network because all algorithms have good quality indicators and no unique best solutions are found for small $n$. On the contrary,

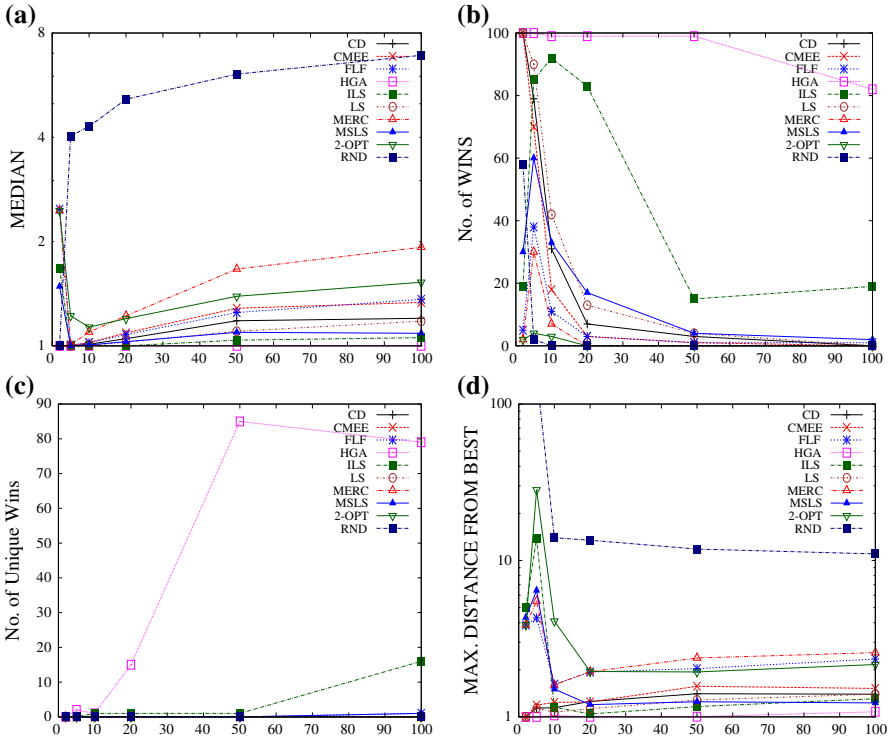


**Fig. 7** Quality versus $|L|$. **a** Median of the relative distance (log scale), **b** number of wins, **c** number of unique wins, **d** relative distance of the worst solutions (log scale)

beyond HGA only ILS, FLF, MSLS are capable of providing unique best solutions when instance sizes are growing (Fig. 6c). Note, that FLF is the only algorithm not based on local-search capable of providing some best solutions for certain instances. Surprisingly, when instance sizes are big ILS builds also some of the worst solutions (Fig. 6d, at $n = 1000$). It happens because ILS depends on the quality of the initial solution. Bad initial solutions can be hard to escape given a runtime limit and big size of the network. Performance of algorithm FLF is diverse. On the one hand, FLF can build very bad solutions (Fig. 6d at $n = 200$). On the other hand, it is able to deliver rare best solutions when instances are big (Fig. 6c, $n = 1000$). On average FLF improves with growing $n$ (Fig. 6a). It applies also to 2-OPT. This demonstrates two features: Firstly, the quality distributions of FLF, or ILS, can stretch beyond the best or the worst end of quality distributions of other algorithms. Thus, they can be both the best, or the worst algorithms, depending on the instances. Secondly, with growing instance size it is increasingly necessary to build longer cycles to visit all lines. Consequently, algorithms guided mainly by length of solutions (2-OPT, FLF) gradually improve solution quality (Fig. 6a).

In Fig. 7 quality indicators are shown against growing number of lines in the network. Surprisingly, algorithms CD, CMEE, LS, have better quality solutions than ILS, MSLS, when the number of lines is small ($|L| = 2, 5$). This phenomenon can be

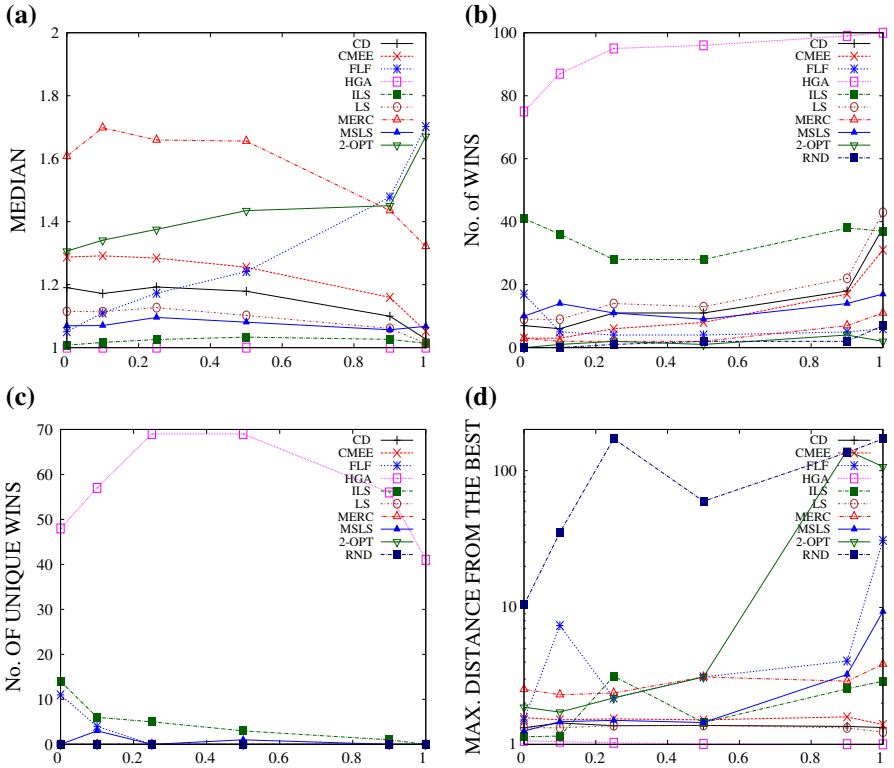


**Fig. 8** Quality versus $\alpha$. **a** Median of the relative distance, **b** number of wins, **c** number of unique wins, **d** relative distance of the worst solutions (log scale)

explained in the following way. When there are only two lines in the network, a simple greedy solution connecting the closest arcs of the two lines is optimum. Algorithms CD, CMEE use this greedy principle and start from each arc or node, respectively. Therefore, they identify a good pair of arcs and build good solutions. Algorithm LS starts from the solution built by CD, thus it is also effective for such instances. It is relatively easy to guess such solutions which is attested by good quality of RND solutions (Fig. 7a, b) for such instances. Algorithms ILS, MSLS were designed with a tacit assumption that there are many lines to connect. Hence, they use elaborate strategies to join the lines ignoring a simple greedy approach. Still, when the number of lines is growing, these algorithms improve their performance which can be seen in instances with $|L| \geq 10$ (Fig. 7a, b). For the instances with even more lines, only algorithms ILS, HGA, and in rare cases FLF, are capable of building best solutions (Fig. 7b, c).

In Fig. 8 solution quality versus probability $\alpha$ of choosing an existing edge by a segment of a line is shown. Algorithm RND is not indicated in Fig. 8a because its median was over 4. For $\alpha$ close to zero the new lines tend to built many new edges, lines are dispersed and only stations are shared. With growing $\alpha$ new lines increasingly use the existing part of the network before constructing any new edges. For $\alpha$ close to 1, the new lines reuse the existing edges before building any new ones and many

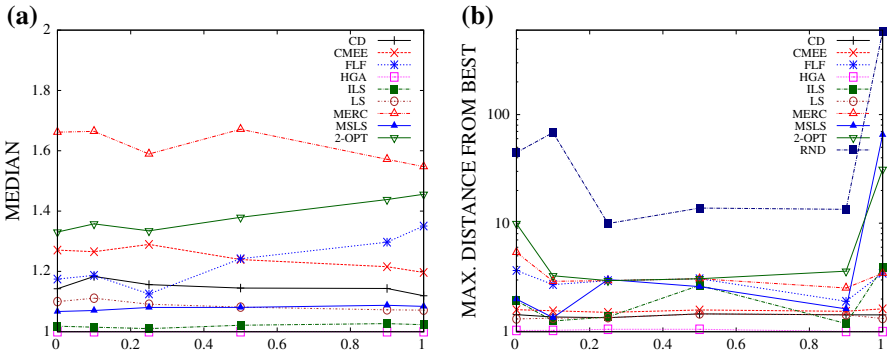


**Fig. 9** Quality versus $\beta$. **a** Median of the relative distance, **b** relative distance of the worst solutions (log scale)

lines share edges. Thus, $\alpha$ is a parameter controlling structure of the subway network, namely, spatial concentration of the lines. It can be seen in Fig. 8a that with growing $\alpha$ solution quality of all algorithms, except for FLF, 2-OPT, is improving on average. This situation can be explained by the concentration of many lines in close edges. Even if some algorithm chooses a sub-optimal set of the edges, they are not very far from the set of edges in the optimum solution. It is also easier to find the best solution for big $\alpha$ (cf. Fig. 8b) such that the number of best solutions found uniquely is decreasing near $\alpha = 1$ (cf. Fig. 8c). It means that for big $\alpha$ good solutions are increasingly a set of the close edges with the big number of colors. All algorithms, but FLF, 2-OPT, have greedy components targeting color number and short distances. Yet, the situation for big $\alpha$ is more diversified because quality distributions are more dispersed (see Fig. 8d). It means that it is both easier to repeat the best solutions, as well as build very bad ones. The worst solution quality deteriorates with $\alpha$ in the case of algorithms ILS, MSLS (Fig. 8d). It is a result of unlucky choice of initial solutions and neighborhood structures lacking greedy arc selection. On the other hand, CMEE, CD are better in this region of $\alpha$ because they greedily choose close arcs with many colors. Moreover, they do it more systematically by restarting search from many initial solutions (arcs or nodes). Performance of FLF, 2-OPT, is affected by growing $\alpha$, so that 2-OPT can be as bad as RND in the worst case (see Fig. 8d, $\alpha = 0.9$). Algorithm FLF is losing, with growing $\alpha$, from one of the best to one of the worst algorithms in quality (Fig. 8a, c). These algorithms were designed with a tacit assumption that the solutions cover big distances. This condition is less and less satisfied with growing $\alpha$. Performance of MERC is getting better with increasing $\alpha$ because MERC is merging the most colorful arcs. However, MERC hardly discerns distances. Therefore, its solutions are worse than, e.g., CMEE's or CD's.

Figure 9 depicts solution quality versus probability $\beta$ of reusing an existing station. For $\beta \approx 0$ a new line builds only new stations once it leaves the existing network. Consequently, networks often have branches comprising only one line. For $\beta \approx 1$ the new line, once it leaves the existing network, is bound to visit an existing station if such a station exists in admissible range of distances and directions. Thus, $\beta$ is one more parameter controlling structural properties of the subway network, namely

degree of line coincidence in the existing stations. Similarly to $\alpha \approx 1$, networks with big $\beta$ are spatially more compact. However, influence of $\beta$ is much smaller than $\alpha$'s because the random process controlled by $\beta$ is more restricted and hence called less frequently. Consequently, quality of the solutions is less susceptible to the changes of $\beta$. This is confirmed in Fig. 9a showing the median of relative distance from the best solution versus $\beta$. The tendencies are similar as in Fig. 8a for parameter $\alpha$, but weaker. We do not show the number of (unique) wins because these quality indicators were not changing noticeably with $\beta$, while HGA, ILS were the best as in the earlier experiments. Only in the worst solutions can any tendency be observed (Fig. 9b). The worst case distance from the best solution, compared with the earlier experiments, is especially big in algorithms 2-OPT, MSLS for $\beta = 1$ and for MERC, 2-OPT, FLF at $\beta = 0$ (see Fig. 9b). For $\beta = 0$ once a line leaves the existing network it never returns and no long-range connections are made. For $\beta = 1$ once a line leaves the existing network it returns at the first opportunity making short parallel branches to the existing network. In both cases it is a good option to choose greedily close colorful arcs as for $\alpha = 1$. However, for $\beta = 0$ or $\beta = 1$ there are edges conveying only one line spread in the network. These edges can be used by MERC, FLF in the initial phase of the algorithms ignoring the most colorful edges in the network. Similar thing happens with MSLS which chooses some bad initial solution and happens to be unable to find a better one by randomized local search.

### 6.3 Advantages in instance shortening

We analyze here the impact of instance shortening described in Sect. 2. Shortened instances have fewer edges and there is a chance that they are solved faster. However, shortening process changes the subway network and the effect can be more unpredictable both on the runtime and the solution quality. The results of instance shortening will be shown as a relative change in the runtime and in the quality of the solutions constructed by a given algorithm. For a set of instances we record ratios of the runtimes on the shortened and on the original instances. Similarly, we calculate the ratio of the solution quality obtained for the shortened instance to the original instance. Values smaller than 1 represent quality improvement or runtime reduction. Quartiles of the ratios calculated for a population of all 2,600 instances are shown in Table 2.

The left part of Table 2 represents quality changes. It can be seen that the first (Q1), and the third (Q3) quartiles are almost always equal to 1. Only for algorithms RND and MSLS can any shift in range [Q1,Q3] be observed. However, the minima and maxima of quality ratios are much affected by instance shortening. It is most visible in algorithms RND, 2-OPT, FLF. This effect could be expected because, as discussed in Sect. 6.2, instance types exist where our algorithms can make a bad decision and perform badly. Algorithms RND, 2-OPT, FLF follow only one path in the solution construction. Consequently, a change in network perception may influence algorithm performance. On the other hand, CD, CMEE, HGA build many solutions and hence are less susceptible to one bad decision.

In the right part of Table 2 the ratio of the time moments when the algorithm provides its best solution is shown. For algorithms HGA, ILS, MSLS the influence of

**Table 2** Effect of instance shortening

| | Relative quality change | | | | | Relative runtime change | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Min | Q1 | Q2 | Q3 | Max | Min | Q1 | Q2 | Q3 | Max |
| 2-OPT | 0.014 | 1.000 | 1.000 | 1.000 | 374.3 | 4.16E-05 | 0.269 | 0.827 | 1.000 | 10.0 |
| CD | 0.896 | 1.000 | 1.000 | 1.000 | 2.678 | 4.01E-05 | 0.372 | 0.888 | 1.000 | 10.0 |
| CMEE | 0.996 | 1.000 | 1.000 | 1.000 | 2.678 | 4.08E-05 | 0.353 | 0.865 | 0.994 | 10.0 |
| FLF | 0.059 | 1.000 | 1.000 | 1.000 | 374.3 | 4.04E-05 | 0.269 | 0.800 | 1.000 | 10.0 |
| HGA | 0.907 | 1.000 | 1.000 | 1.000 | 2.678 | 0.998 | 1.000 | 1.000 | 1.000 | 1.002 |
| ILS | 0.374 | 0.999 | 1.000 | 1.001 | 394.0 | 0.692 | 1.000 | 1.000 | 1.000 | 1.092 |
| LS | 0.818 | 1.000 | 1.000 | 1.000 | 2.678 | 4.01E-05 | 0.368 | 0.887 | 1.000 | 10.0 |
| MERC | 0.315 | 1.000 | 1.000 | 1.000 | 374.3 | 4.16E-05 | 0.261 | 0.777 | 1.000 | 10.0 |
| MSLS | 0.107 | 0.985 | 1.000 | 1.016 | 5.838 | 0.852 | 1.000 | 1.000 | 1.000 | 1.168 |
| RND | 0.002 | 0.937 | 1.000 | 1.088 | 742.8 | 4.17E-05 | 0.236 | 0.736 | 1.000 | 10.0 |

instance shortening is small. These algorithms finish their computation by reaching the time limit and keep providing solutions in the whole allowed runtime. Shortening has little influence on their behavior. Contrarily, runtime of one-pass algorithms like CD, CMEE, MERC, FLF is reduced by instance shortening. The first quartile of runtime ratios is roughly 30 % of the original runtime. In some cases, the runtime can change more extremely. It can increase by an order of magnitude and decrease by over four orders of magnitude. A more detailed analysis reveals that the parameters affecting reduction in the runtime are: number of stations $n$, number of lines $|L|$, and probability of choosing an existing edge $\alpha$. With growing $n$ fraction of stations on long chains is growing. Hence, reducing them to several edges reduces the runtime. The two remaining factors correlate with the types of instances particularly hard for certain algorithms. Shortening effects are visible if the number of lines is small, i.e. $|L| = 2, 5$. Changing the networks by shortening them reduces chances of making bad initial decisions, and then solution quality of almost all algorithms improves. With growing $|L|$ this effect disappears. Similar phenomenon arises for $\alpha = 1$, when lines reuse existing edges, and hence are very concentrated. As discussed in Sect. 6.2, algorithms like 2-OPT, FLF, tend to ignore good greedy solutions. Shortening instances at $\alpha = 1$ reduce chance of making such bad choices, and quality of the solutions improves.

It can be concluded that the overall effect of instance shortening is positive. On average, the changes in quality are limited, and the runtime is reduced. However, in rare cases instance shortening may severely deteriorate solution quality or runtime.

### 6.4 Performance on real instances

In this section we report on performance of our algorithms on the real city instances introduced in Drozdowski et al. (2012). Parameters of the instances are shown in Table 3. The last column in Table 3 comprises uMTG cycle lengths obtained by a

**Table 3** Real instances (Drozdowski et al. 2012)

| Name | $n$ | $m$ | $\|L\|$ | uMTG length |
|---|---|---|---|---|
| Beijing | 126 | 276 | 9 | 37* |
| Berlin | 175 | 372 | 10 | 32* |
| Krakow-tram | 155 | 368 | 27 | 22* |
| Mexico | 172 | 384 | 13 | 55 |
| Moskovskij | 148 | 342 | 12 | 37 |
| Paris | 338 | 804 | 21 | 51 |
| Poznan-tram | 116 | 258 | 19 | 15* |
| Seoul | 393 | 878 | 11 | 70 |
| Shanghai | 200 | 450 | 12 | 41 |
| Stuttgart-tram | 179 | 380 | 14 | 37* |
| TokyoSub | 205 | 502 | 13 | 19* |
| Tube | 337 | 780 | 13 | 18* |

The known optima are marked with an asterisk

branch-and-bound algorithm with 100 hour runtime limit. The known optima are marked with an asterisk in Table 3.

Results of our algorithms on the above instances obtained in at least 10 runs with 600 s time limit are shown in Table 4. In most of the cases solution values were equal in all runs. For such cases we provide the only solution value in Table 4. Where solution values differed minimum, median, and maximum of the objective function are provided. It can be seen in Table 4 that the best solution of Paris instance has been improved by algorithms HGA,ILS from 51 to 47. All the best known solutions were found by some of our algorithms. Algorithms HGA and ILS were able to find these best solutions at least in some runs. Algorithm MSLS was the third best, because it was able to repeat 8 best solutions out of 12 in some runs. Algorithm MERC, and not surprisingly RND, turned out to be the worst because they were not able to find any of the best known solutions. The RND method constructs solutions which are very far both from the best known solutions, and from the solutions constructed by our algorithms. Thus, it can be concluded that our algorithms build good solutions not as a result of simple nature of the considered problem. Randomized algorithms HGA, ILS, MSLS build solutions with little dispersion of the objective function value. This can be considered as a sign of their robustness. Overall, performance of HGA and ILS on the real instances can be regarded as very good.

## 7 Conclusions

In this paper we proposed a set of heuristics for a problem of finding the shortest cycle visiting all lines of a subway network. The heuristics may be divided into two main groups: one-pass algorithms and local-search based algorithms. In order to compare performance of so different types of methods a novel approach has been proposed. The diagrams of solution quality versus time introduced in Sect. 6.1 demonstrate that the algorithms differ in the trade-off between solution quality and runtime cost. It was possible to determine that some heuristics constructing good solutions in a long

**Table 4** Solution quality for real instances

| Name | Best so far | 2-OPT | CD | CMEE | FLF | HGA | ILS | LS | MERC | MSLS | RND |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Beijing | 37* | 45 | 41 | 39 | 39 | 37 | 37 | 41 | 52 | 37, 39, 39 | 90, 107, 141 |
| Berlin | 32* | 32 | 32 | 34 | 33 | 32 | 32 | 32 | 41 | 32, 32, 34 | 154, 163, 182 |
| Krakow-tram | 22* | 50 | 24 | 28 | 44 | 22 | 22, 22, 23 | 24 | 26 | 24, 26, 27 | 146, 181, 191 |
| Mexico | 55 | 66 | 62 | 65 | 59 | 55 | 55 | 59 | 67 | 55, 57, 58 | 203, 226, 232 |
| Moskovskij | 37 | 39 | 39 | 43 | 40 | 37 | 37 | 39 | 45 | 37, 38, 39 | 136, 151, 169 |
| Paris | 51 | 62 | 57 | 62 | 56 | **47** | **47**, 48, 48 | 51 | 71 | 48, 50, 53 | 233, 240, 284 |
| Poznan-tram | 15* | 34 | 15 | 15 | 28 | 15 | 15, 15, 16 | 15 | 22 | 15, 19, 22 | 66, 114, 117 |
| Seoul | 70 | 89 | 72 | 75 | 76 | 70 | 70 | 70 | 96 | 70,70,71 | 305, 335, 379 |
| Shanghai | 41 | 51 | 45 | 47 | 41 | 41 | 41 | 41 | 46 | 41, 43, 43 | 144, 151, 168 |
| Stuttgart-tram | 37* | 57 | 39 | 39 | 50 | 37 | 37 | 37 | 56 | 37, 37, 40 | 116, 184, 197 |
| TokyoSub | 19* | 30 | 22 | 24 | 23 | 19, 19, 20 | 19 | 22 | 34 | 21, 23, 24 | 180, 208, 230 |
| Tube | 18* | 36 | 21 | 25 | 29 | 18 | 18 | 21 | 26 | 20, 21, 26 | 107, 146, 184 |

Asterisks mark known optima. New best solutions are shown in boldface

run, are dominated by some simpler methods that give lower quality solutions but much earlier. For example, hybrid genetic algorithm (HGA) builds best solutions, but this quality comes with a price. Moreover, by comparison of the quality versus time diagrams at different time snapshots, it was possible to trace evolution of the quality-time trade-off. To further analyze which features of the instances determine solution quality a second set of experiments was performed. By sweeping along a range of values in some analyzed parameter it was possible to examine its influence on the quality of generated solutions.

The analysis revealed that our algorithms have some presumptions on the form of a solution. For example, algorithm FLF tacitly assumes that the MTG cycle covers big parts of the subway graph and hence is inevitably long. Thus, FLF starts by connecting two furthest lines with the expectation that later incorporating the other lines will be less costly. Algorithms CMEE, CD assume that a solution comprises a set of closely located edges. Algorithm MERC is built with an assumption that lines are spatially clustered. Local search-based algorithms like MSLS, ILS assume that the solution is obtained by extensive search of some combinatorial solution space. Obviously, there are instances that satisfy such assumptions and other that do not. The above conclusions were achieved by a particular design of computational experiments. We believe that both types of experiments: on quality versus time and on sweeping one parameter range while randomizing the remaining ones can be repeated for other combinatorial optimization problems.

**Acknowledgments** This research has been partially supported by a Grant of Polish National Science Center.



## References

Abdullah, S., Turabieh, H., McCollum, B., McMullan, P.: A hybrid metaheuristic approach to the university course timetabling problem. J. Heuristics **18**, 123 (2012)

Arkin, E., Hassin, R.: Approximation algorithms for the geometric covering salesman problem. Discrete Appl. Math. **55**, 197–218 (1994)

Black, P.E.: Chinese postman problem. In: Dictionary of Algorithms and Data Structures, US National Institute of Standards and Technology. http://www.nist.gov/dads/HTML/chinesePostman.html (2012). Accessed 06 December 2012

Berlińska, J., Drozdowski, M., Lawenda, M.: Experimental study of scheduling with memory constraints using hybrid methods. J. Comput. Appl. Math. **232**, 638–654 (2009)

Codenotti, B., Manzini, G., Margara, L., Resta, G.: Perturbation: an efficient technique for the solution of very large instances of the Euclidean TSP. INFORMS J. Comput. **8**, 125–133 (1996)

Croes, G.A.: A method for solving traveling-salesman problem. Oper. Res. **6**, 791–812 (1958)

Current, J., Schilling, D.: The covering salesman problem. Trans. Sci. **23**, 208–213 (1989)

Drozdowski, M., Kowalski, D., Mizgajski, J., Mokwa, D., Pawlak, G.: Mind the gap: a study of tube tour. Comput. Oper. Res. **39**, 2705–2714 (2012)

Eiselt, H.A., Gendreau, M., Laporte, G.: Arc routing problems, part II: the rural postman problem. Oper. Res. **43**, 399–414 (1995)

Eulero, L.: Solutio problematis ad geometriam situs. Commentarii Academiae Scientiarum Petropolitanae **8**, 128–140 (1741)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H.Freeman and Co, San Francisco (1979)
Laporte, G., Riera-Ledesma, J., Salazar-González, J.: A branch-and-cut algorithm for the undirected traveling purchaser problem. Oper. Res. **51**, 940–951 (2003)
Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**, 498–516 (1973)
Martin, O., Otto, S.W., Felten, W.: Large step Markov chains for the TSP incorporating local search heuristics. Oper. Res. Lett. **11**, 219–224 (1992)
Safra, S., Schwartz, O.: On the complexity of approximating TSP with neighborhoods and related problems. Comput. Complex. **14**, 281–307 (2006)
Talbi, E.G.: A taxonomy of hybrid metaheuristics. J. Heuristics **8**, 541–564 (2002)
Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: a survey. Eur. J. Oper. Res. **209**, 110 (2011)