Universität Dortmund
Wirtschafts- und Sozialwissenschaftliche Fakultät

Diskussionsbeiträge des Fachgebietes
Operations Research und Wirtschaftsinformatik
No. 30 (2005)

On variants of the k-Chinese Postman Problem

André Osterhues
Frank Mariak

Universität Dortmund, Operations Research und Wirtschaftsinformatik, Vogelpothsweg 87, D 44 221 Dortmund, Germany

On variants of the k-Chinese Postman Problem

André Osterhues[*] Frank Mariak[†]
Operations Research and Wirtschaftsinformatik
University of Dortmund
D-44221 Dortmund
Germany

August 31, 2005

Abstract

The k-Chinese Postman Problem (k-CPP) states that $k \geq 2$ postmen have to service a given graph, where each edge has a service time and a cruise time. The task is to find a k-postman tour with minimal total time. In practice, this often leads to unsatisfactory results, as differences in total time for each postman are not accounted for.

Besides presenting the Minimum Absolute Deviation (MAD) k-CPP (also known as Balanced k-CPP), in this paper two novel variants of the k-CPP, the Minimum Square Deviation (MSD) k-CPP and the Minimum Overtime (MOT) k-CPP are introduced. The deviation is the sum of the absolute differences between target time and actual time, added up over all k postmen. For the MAD k-CPP, the deviation from the allowed time is minimized, while for the MSD k-CPP, the square difference is added up, thus penalizing higher deviations. For the MOT k-CPP, only time exceeding the allowed time is considered.

New heuristics for solving the k-Chinese Postman Problem and its variants are presented and detailed results on standard test data are given.

1 Introduction

The k-Chinese Postman Problem (k-CPP) originates from the Chinese Postman Problem (Guan [1]), in which a single postman is required to service a number of streets from a post office. This problem can naturally be modeled by a graph in which streets are abstracted by edges and vertices represent crossings, resp. the post office. The objective is to minimize the total tour cost.

In order to find a finite solution of minimum cost, the graph needs to be connected and all edge costs[1] have to be non-negative. Each connected graph has an even number of vertices with odd degree. To make a graph Eulerian and find a closed walk traversing each

[*]mail@andreosterhues.de

[†]F.Mariak@t-online.de

[1]The terms "time" and "costs" are used synonymously throughout this paper.

edge at least once, edges must be added to connect vertices with odd degree. To change the graph so that each vertex has even degree, a minimum-weight perfect matching is calculated and an optimal solution to the CPP is available. This problem can be solved by a polynomial time algorithm as first described by Edmonds and Johnson [2].

A problem related to the CPP is the Rural Postman Problem (RPP). While the CPP implies that all edges have to be serviced, the Rural Postman problem as introduced by Orloff[3] addresses that some real world problems only require a subset of edges to be serviced. The edges are split into a set of required and non-required edges. A closed walk of minimum cost within the graph has to be found, traversing the set of required edges exactly once. The RPP has been proven to be $\mathcal{NP}$-hard by Orloff [3]. Assuming $\mathcal{NP} \neq \mathcal{P}$, an exact solution can not be derived in polynomial time.

There is a number of real world problems related to the k-CPP. Usually a company does not employ a single person but several persons who cover the service for a subproblem. There are services like post delivery, parcel services, snow clearence services or milk delivery which require several persons to provide service at minimum cost in a limited time. The k-Chinese Postman Problem (k-CPP) is derived from the CPP by using $k \geq 2$ postmen with the objective to minimize the sum of all tour costs.

From an economical point of view, minimizing the sum of all tour costs for several postmen as described in the k-CPP is not sufficient. In the worst case, one postman has to service all edges while all other postmen just walk out of the depot and return right away. Therefore, routes of similar length for all postmen are desired. In the past Frederickson et al. [4] and Ahr [5] proposed the Minimum-Maximum (MM) k-CPP. Instead of minimizing the sum of all tour costs, the objective function for the MM k-CPP is to minimize the cost of the tour with the maximum cost. It was shown to be $\mathcal{NP}$-hard, but a heuristic with a $(2 - 1/k)$ approximation can be deployed. Even though the problem was first described over 25 years ago, routing tasks related to the MM k-CPP have been barely studied in the literature.

Based on the Balanced k-CPP (Degenhardt [6]), which is called Minimum Absolute Deviation (MAD) k-CPP throughout this paper, we present two novel objective functions for the k-CPP, all using a pre-defined allowed time for each postman. Instead of minimizing the maximum tour cost like in the MM k-CPP, we define the Minimum Square Deviation (MSD) k-CPP and the Minimum Overtime (MOT) k-CPP. The MAD k-CPP objective function minimizes the sum of all deviations from an allowed service time for a single postman. The MSD k-CPP function sums the squares of the absolute deviations to reduce peaks in service times for all postmen. The MOT k-CPP aims at minimizing the sum of overtime for all postmen, though not penalizing working times smaller than the allowed time.

2 Problem formulation

This section is organized as follows: First, general terminology used throughout this paper is described. Then, several problems related to the solution are formulated. Problems formulated are the Chinese Postman Problem (CPP) with seperated service and cruise edges, the k-CPP adding additional postmen, followed by the Minimum-Maximum k-CPP and variants of it using different objective functions.

2.1 Terminology

This section introduces the general terminology used throughout this paper in the following sections. When more specific terminology is used, it will be explained in the related paragraph. All introduced problems are described by an undirected graph $G = (V, E)$ which consists of a finite nonempty set V of vertices and a finite set E of edges. An unordered pair of vertices (v_i, v_j) is associated with every edge. Throughout this paper, two basic types of edges are used. Whenever a finite set S is defined, it refers to edges with service costs $s : S \rightarrow \mathbb{R}_0^+$ (service edges), while a finite set C refers to edges with cruise costs $c : C \rightarrow \mathbb{R}_0^+$ (cruise edges).

In a graph G, a walk is a finite sequence of vertices $W = (v_1, \ldots, v_k), k \geq 1$. A walk may also be considered as a sequence of its traversed edges, $W = (e_1, e_2, \ldots, e_k)$ with $k \geq 1$. A walk is called closed if it has nonzero length and its source and destination vertex are identical. It will also be called tour T later on. With $cost(T) = \sum_{e \in T \cap S} s(e) + \sum_{e \in T \cap C} c(e)$ we denote the costs of the tour T, i.e. the sum of the service resp. cruise costs of the tour edges.

A walk in which all vertices are distinct is called path P. The shortest path $SP(v_i, v_j)$ is defined between two vertices v_i and v_j where $i, j \in \{1, \ldots, k\}$ with source v_i and destination v_k, spanning the shortest possible distance between those vertices.

A walk that traverses every edge of a graph G exactly once is called Eulerian trail. In analogy to a closed walk we refer to a closed Eulerian trail as an Eulerian tour.

A matching M in a graph $G = (V, E)$ is a set of edges such that no two edges of M have a common vertex and it is called a perfect matching M if every vertex v_i is contained in one edge of M. G is called connected if there exists a path from v_i to v_j for all vertices $v_i, v_j \in V$ with $v_i \neq v_j$.

2.2 CPP

The Chinese Postman Problem was first formulated by Guan [1]. An optimal solution to the problem can be found by making the related graph Eulerian and then finding a tour traversing each edge once. EulerianTour (Algorithm 1) outlines how the Eulerian tour is calculated.

Algorithm 1 EulerianTour

Input: An Eulerian graph $G = (V, E)$ and the depot vertex v_1.
Output: A tour T covering all edges in E.

```
1: Insert $v_1$ into tour $T$, mark $v_1$ as visited and mark all edges as not traversed.
2: while not all edges traversed do
3:     Find cycle starting at an already visited vertex $v_i$.
4:     Insert cycle at first occurence of $v_i$ in tour $T$.
5:     Mark all vertices of cycle as visited and all edges of cycle as traversed.
6: end while
7: Output $T$.
```

A CPP instance is defined by a connected undirected Graph $G = (V, (S, C))$ and edge costs $s : S \rightarrow \mathbb{R}_0^+$, $c : C \rightarrow \mathbb{R}_0^+$. The objective is to find a closed walk T^* in G with minimum costs traversing each edge from edge set S once, e.g. to find a single postman tour T^* with $cost(T^*) \rightarrow min$. ChinesePostmanProblem (Algorithm 2) outlines the calculation of the tour which covers all edges in S. For the calculation of the minimum-weight perfect matching in line 4, the $O(|V|^3)$ algorithm of Edmonds and Johnson [2] can be used.

Algorithm 2 ChinesePostmanProblem

Input: A connected graph $G = (V, (S, C))$ and the depot vertex v_1.
Output: A tour T covering all edges in S.

```
1: Let $V_{odd} \subset V$ be the set of vertices with odd degree.
2: Calculate shortest paths $SP(v_i, v_j)\ \forall v_i, v_j \in V_{odd}$ on edges from set $C$.
3: Let $\widetilde{G} = (V_{odd}, \{SP(v_i, v_j) \mid v_i, v_j \in V_{odd}\})$.
4: Calculate a minimum-weight perfect matching $M \subset C$ on the graph $\widetilde{G}$.
5: $T = \text{EulerianTour}(G', v_1)$ with $G' = (V, E')$ and $E' = S \cup M$.
6: Output $T$.
```

2.3 k-CPP

The k-Chinese Postman Problem (k-CPP) originates from the Chinese Postman Problem (CPP). Real world problems require the deployment of several postman and not just one.

A k-CPP instance is defined by $k \geq 2$ postmen on a connected undirected graph $G = (V, (S, C))$, a distinguished depot vertex v_1, a service time $s : S \rightarrow \mathbb{R}_0^+$ and a cruise time $c : C \rightarrow \mathbb{R}_0^+$. The postmen have to walk all service edges in S with a service time s, hereby called service edges, at least once.

$$\sum_{i=1}^{k} cost(T_i) \quad \rightarrow min \qquad k\text{-CPP} \tag{1}$$

2.4 MM k-CPP

The Minimum-Maximum k-Chinese Postman Problem (MM k-CPP) can be described by a connected unidirected graph, a distringuished depot vertex v_1, a fixed number of postmen which have to walk a set of weighted edges (with a specified service time) like in the k-CPP. Now, a tour with a minimum sum of all tour costs is not the objective. Instead, minimizing the cost of the postman tour with the maximum cost is desired.

$$\max_{i=1}^{k}\{cost(T_i)\} \quad \rightarrow min \qquad \text{MM } k\text{-CPP (Minimum-Maximum)} \tag{2}$$

A better distribution of working times across all postmen is the objective of this modified k-CPP.

2.5 Variants of the k-CPP

Besides the two latter problems (k-CPP and MM k-CPP), this paper describes three additional objective functions. The first of these (the MAD k-CPP) has been introduced by Degenhardt [6] (although called Balanced k-CPP in his lecture), while the other two — to the best of our knowledge — have not yet been covered in the literature.

2.5.1 MAD *k*-CPP

The MAD *k*-CPP objective function tries to minimize the sum of all deviations from an allowed service time for a single postman.

$$\sum_{i=1}^{k} |cost(T_i) - l| \quad \rightarrow min \qquad \text{MAD } k\text{-CPP (Minimum Absolute Deviance)} \tag{3}$$

This aims at fitting the working time of all postmen to a specified allowed time closer as known by the MM *k*-CPP, resulting in similar working times for every postman.

2.5.2 MSD *k*-CPP

The MSD *k*-CPP function tries to reach even more similar services times for all postmen.

$$\sum_{i=1}^{k} (cost(T_i) - l)^2 \quad \rightarrow min \qquad \text{MSD } k\text{-CPP (Minimum Square Deviance)} \tag{4}$$

In order to penalize higher deviations and to reduce peaks in working times for single postman, the square difference is added up, resulting in a more evenly distributed working time across all postmen.

2.5.3 MOT *k*-CPP

The MOT *k*-CPP aims at minimizing the sum of overtime for all postmen.

$$\sum_{i=1}^{k} max\{cost(T_i) - l, 0\} \quad \rightarrow min \qquad \text{MOT } k\text{-CPP (Minimum Overtime)} \tag{5}$$

While a similar distribution of working times is desirable, minimizing overtime is a real world objective as well. The MOT *k*-CPP does not penalize working times smaller than the allowed time, thus the deviation of these times is not minimized (similar to the MM *k*-CPP). If the allowed time is specified too high, using the MOT *k*-CPP can lead to improper results.

3 Solution Algorithms

The *k*-CPP is $\mathcal{NP}$-hard, which can be proven by a reduction to *k* RPPs. Thus, if an exact solution is desirable, an integer programming method like Branch and Bound can be utilized. Otherwise, if suboptimal solutions are permittable, the problem can be solved by a heuristic method. In our scenario, we choose a Branch and Bound method.

This section is organized as follows: First, our approach to allocate the service edges to the postmen is introduced, which leads to the RPP. Then, a heuristic for solving the RPP is presented. Finally, the Branch and Bound method is examined in detail.

3.1 Longest Processing Time

The Longest Processing Time (LPT) scheduling algorithm was introduced and analyzed by Graham [7]. The objective is to allocate r independent tasks to n identical processing units such that the makespan ω (the total time required to finish all tasks) is minimized. The LPT scheme first sorts the tasks in descending order according to the time they consume. Then, each task is allocated to the processing unit which currently has the smallest load. This algorithm guarantees a worst-case approximation factor of

$$\frac{\omega_{LPT}}{\omega_{OPT}} \leq \frac{4}{3} - \frac{1}{3n}.$$

To obtain a heuristic to solve the k-CPP, we propose to schedule service edges to postmen using LPT. However, to take the distance of an edge from the depot vertex into account, not only the cost of traversing one service edge should determine its LPT cost, but also the traversal cost from and to the depot vertex, which is the cost of the shortest path. Hence, the LPT cost of an edge $s \in S$ is defined as follows:

$$LPTcost(s) = cost(SP(v_1, v_i)) + s(v_i, v_j) + cost(SP(v_j, v_1)).$$

Each postman's current resp. total tour cost can easily be calculated as the sum of the LPT cost of the edges he has to service.

The LPT scheduling algorithm assures that all postmen have walks of similar length. However, after processing each service edge the postman returns to the depot instead of using a (possibly) shorter path to the next service edge. Therefore, the current tour cost calculation in the LPT algorithm should be modified in order to reflect such shorter paths. Also, if the postman does not return to the depot vertex each time, the order in which the service edges are being processed has a crucial influence on the tour cost and should be considered as well. In fact, the problem of having a set of service edges and finding an otimal tour covering these edges is exactly the Rural Postman Problem (RPP), which will be presented in more detail in the following section.

3.2 Rural Postman Problem

For edge sets consisting of only one edge, the optimal tour is a traversal from the deposit to the service edge and back. For two service edges, there are four possibilities and in general for n edges there are $n! \cdot 2^{n-1}$ possibilities, thus rendering a brute-force approach infeasible.

A simple heuristic to solve the RPP is to connect the disjoint components of the graph using shortest paths (using Floyd's well-known algorithm [8]) and then solve the 1-CPP on the resulting graph. A component consists either of a set of vertices connected by service edges or only the depot vertex. Algorithm 3 depicts the ConnectGraph algorithm in pseudo-code. The algorithm uses the greedy paradigma to add the shortest path between two disjoint components (cf. line 5) in order to connect them. This is iterated until there is only one component, which means that there is a spanning tree covering all service edges and the depot vertex.

RuralPostmanProblem is shown in Algorithm 4. Edges added to connect the graph have to be marked as service edges (however, with cruise edge costs) to the 1-CPP solver (cf. line 2), to ensure that they are part of the tour. After the 1-CPP tour has been

Algorithm 3 CONNECTGRAPH

```
Input:  A graph $G = (V, (S, C))$ and the depot vertex $v_1$.
Output: A set of connecting edges $\widehat{C}$.
1: Calculate shortest paths $SP(v_i, v_j)\ \forall\, i, j \in \{1, \dots, n\}$ on edges from set $C$.
2: Let $\widehat{C} = \emptyset$.
3: while $G' = (V', E')$ with $V' = \{v | v \in S \vee v \in \widehat{C}\}$ and $E' = S \cup \widehat{C}$ is not connected
   and $v_1 \notin V'$ do
4:     Find minimum-cost $SP(v_i, v_j)$ with $v_i$ and $v_j$ in different components.
5:     Add the edges in $SP(v_i, v_j)$ to $\widehat{C}$.
6: end while
7: Output $\widehat{C}$.
```

computed, they must be readjusted to cruise edges (cf. line 6). This way, a single tour can be calculated. In practice, Algorithm 4 often yields optimal results.

Algorithm 4 RURALPOSTMANPROBLEM

```
Input:  A graph $G = (V, (S, C))$ and the depot vertex $v_1$.
Output: A tour $T$ covering all edges in $S$.
1: $\widehat{C}$ = CONNECTGRAPH($G, v_1$).
2: Set $G' = (V, (S', C))$ with $S' = S \cup \widehat{C}$.
3: $T$ = CHINESEPOSTMANPROBLEM($G', v_1$).
4: for all edges $e \in T$ do
5:     if $e \in \widehat{C}$ then.
6:         Change edge type from service to cruise.
7:     end if
8: end for
9: Output $T$.
```

3.3 Branch and Bound

To solve the variants of the k-CPP, a Branch and Bound strategy was used. Each problem instance consists of a lower bound, an upper bound and a mapping of service edges to postmen. The first f service edges are fixed, i.e. their mapping remains unchanged for all subsequent subproblems. For the first problem instance, only edge s_1 is fixed to postman 1. All problem instances constitute a Branch and Bound tree. Because for further iterations only tree leaves have to be considered, the Branch and Bound tree can be implemented using a priority queue data structure.

3.3.1 Branching

In each iteration, the instance with the currently smallest lower bound (which is first in the priority queue) is processed in the following way. First, the branching edge s_{f+1} is determined from the sorted set S of service edges. Then, $\phi = \min(f+1, k)$ subproblems are created by allocating the branching edge s_{f+1} to postman i with $i = 1, \ldots, \phi$ and turning it into a fixed edge. The remaining edges are scheduled using LPT for all subproblems and the resulting lower and upper bounds are calculated. Finally, the subproblems are inserted into the priority queue according to their lower bound value.

3.3.2 Lower bound for variants of the k-CPP

The lower bounds used for solving the MM k-CPP such as the Node Duplication Lower Bound (NDLB, cf. Ahr [5]) cannot be used for the three novel variants of the k-CPP because they rely on making assumptions about the maximum tour only. In our case, the costs of all tours have to be considered. Therefore, a rather simple but efficient lower bound was used: for a given problem instance, only the service edges which are fixedly assigned are considered. For these, only the tour costs exceeding the allowed time are accounted for. Summed up over all postmen, this represents a valid lower bound — assumed the RPP is always solved to optimality. In our implementation a heuristic is used to solve the RPP and therefore sometimes suboptimal lower bounds are obtained.

3.3.3 Upper bound for variants of the k-CPP

As an upper bound heuristic, the LPT scheduling as described above is used. Algorithm 5 shows the ScheduleEdges algorithm. First, the service edges are sorted according to their LPT cost (cf. lines 1–4). In the main loop, the current service edge is scheduled to the postman i with the current least load (line 11). Then, this postman's tour is updated using the RuralPostmanProblem (line 12). Finally, the new postman i^* with the least tour cost is identified (line 14).

Algorithm 5 ScheduleEdges

Input: A graph $G = (V, (S, C))$, the depot vertex v_1 and the number of postmen k.
Output: k tours $T_1, \ldots, T_k$ covering all edges in S.

```
 1: for all edges $s = (v_i, v_j) \in S$ do
 2:     Calculate $LPTcost(s) = cost(SP(v_1, v_i)) + s(v_i, v_j) + cost(SP(v_j, v_1))$.
 3: end for
 4: Sort $S$ in descending order according to $LPTcost(s)$.
 5: for $i \leftarrow 1, \ldots, k$ do
 6:     Set $S_i = \emptyset$.                    ▷ $S_i$ : postman $i$'s set of service edges
 7:     Set $T_i = \emptyset$.
 8: end for
 9: Let $i \leftarrow 1$.                                          ▷ $i$ : current postman
10: for all edges $s \in S$ do
11:     Add $s$ to $S_i$.
12:     $T_i$ = RuralPostmanProblem($G'$) with $G' = (V, (S_i, C))$.
13:     Calculate $cost^* = \min_{i=1}^{k}\{cost(T_i)\}$.
14:     Set $i \leftarrow i^*$ with $cost(T_{i^*}) = cost^*$.
15: end for
16: Output $T_1, \ldots, T_k$.
```

Objective function	MAD				MSD				MOT				CPP
Postman	1	2	3	$\sum$	1	2	3	$\sum$	1	2	3	$\sum$	1
Service time	28	26	21	75	21	28	26	75	22	29	24	75	75
Cruise time	5	4	9	18	10	4	4	18	7	6	2	15	9
Total time	33	30	30	93	31	32	30	93	29	35	26	90	84
Allowed time	29	29	29	87	29	29	29	87	29	29	29	87	–
Deviation	4	1	1	6	4	9	1	14	0	6	0	6	–

Table 1: Results

4 Results

In this section, some results for the three different variants of the k-CPP are presented. Table 1 shows the results for the simple graph depicted in Figure 1 (a).

Comparing the total time of the MAD and MSD functions, it is noticeable that both objective functions give identical results (93) while for the MSD the maximum deviation for all postman is smaller (maximum total time: 32) than for the MAD (maximum total time: 33). So it gives more balanced results for every postman, which is the desired effect. For this example, the MOT function could reduce the total time of all postmen from 93 (MAD and MSD) to 90. However, at the cost of one postman working significantly more (35) and one postman working less (26) than the allowed time (29). Compared to the 1-CPP (shown in the last column), the MOT function achieves the closest result for this example, with only two additional cruise edges (the two dashed cruise edges between vertices 1 and 4 in Figure 1 (d)).

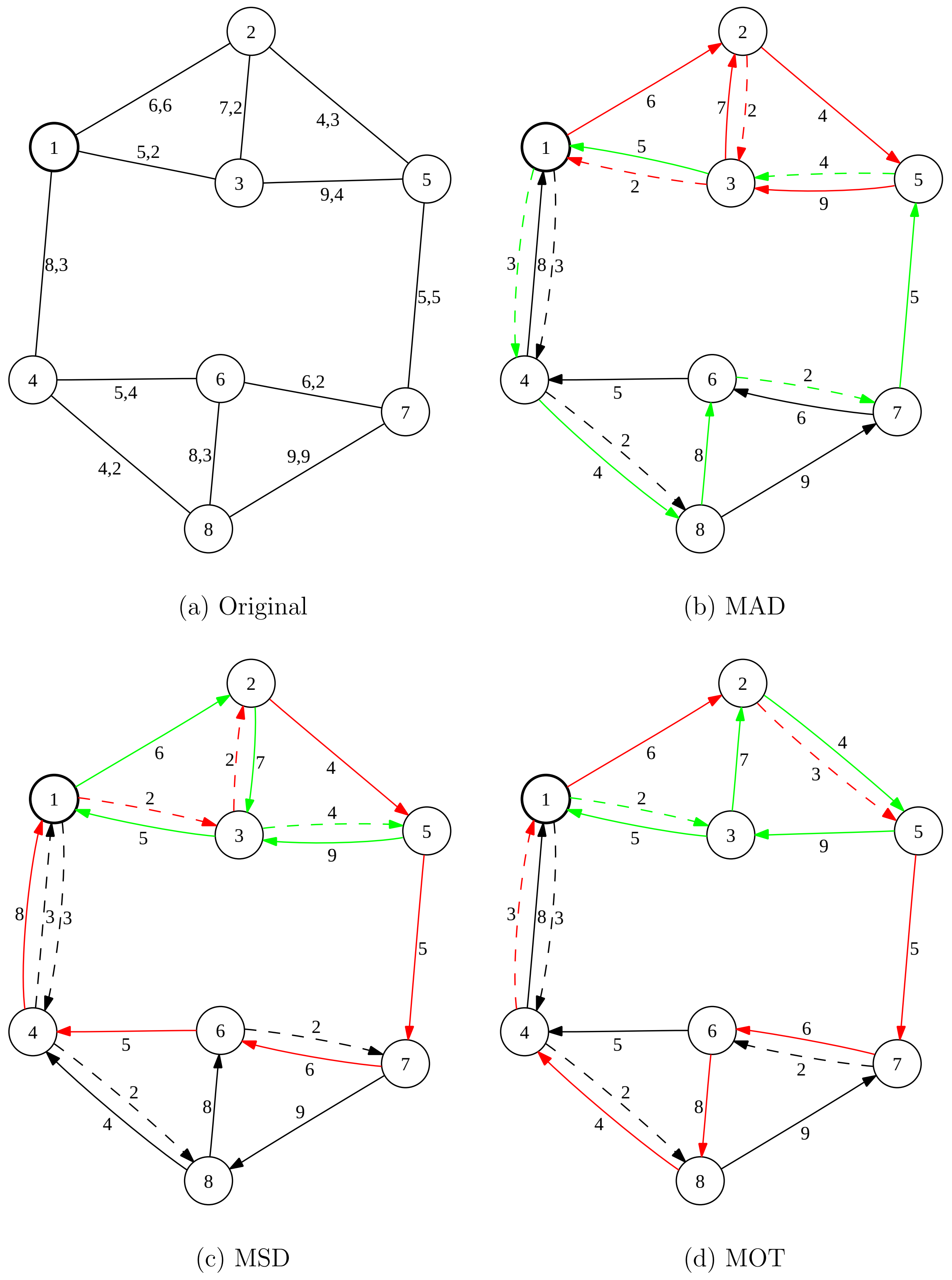
Solving the variants of the k-CPP to optimality for the example graph took only seconds on a 2 GHz Pentium-4 computer with each objective function requiring about 2000 iterations. Further experiments with larger graphs point out that a (near-)optimal solution is obtainable within reasonable time for graphs composed of about 25 edges. Also, results on larger graphs correspond approximately to those of the small example graph.

5 Conclusion

Three variants of the k-CPP (two of which are novel) have been introduced and analyzed. For real life data, the results are more economically reasonable than e.g. those of the k-CPP and the MM k-CPP.

For small problems (graph with less than about 25 edges), the results appear to be optimal or near-optimal and can be calculated in moderate time (i.e. a couple of hours on modern hardware).

The presented algorithm only uses a heuristic to solve the RPP, hence it does not always achieve optimum results. However, by using exact algorithms (such as the Branch and Bound method) to solve the RPP, the k-CPP could be solved to optimality. As this has not (yet) been implemented, further research has to be pursued to judge whether the presented algorithm achieves optimal or near-optimal results also for larger problem instances.



(a) Original

(b) MAD

(c) MSD

(d) MOT

Figure 1: Example for the variants of the k-CPP. (a) The original graph. Each edge is labeled with service and cruise costs. (b) The tours for 3 postmen and an allowed time of 29 using the MAD k-CPP objective function, (c) using the MSD k-CPP objective function and (d) using the MOT k-CPP objective function. Service edges are solid and cruise edges are dashed.

Acknowledgements

The authors thank Jan Degenhardt for constructive comments and assistance.

References

[1] M.-K. Guan, "Graphic programming using odd or even points," *Chinese Mathematics*, vol. 1, no. 3, pp. 273–277, 1962.

[2] J. Edmonds and E. L. Johnson, "Matching, Euler tours and the Chinese postman," *Mathematical Programming*, vol. 5, pp. 88–124, 1973.

[3] C. S. Orloff, "A Fundamental Problem in Vehicle Routing," *Networks*, vol. 4, pp. 35–64, 1974.

[4] G. N. Frederickson, M. S. Hecht, and C. E. Kim, "Approximation Algorithms for Some Routing Problems," *SIAM Journal of Computing*, vol. 7, pp. 178–193, May 1978.

[5] D. Ahr, *Contributions to Multiple Postmen Problems.* Ph.D. thesis, University of Heidelberg, September 2004.

[6] J. Degenhardt, "An Ant-Algorithm for the balanced k-Chinese Postmen Problem," in *Operations Research Conference 2004 (OR '04)*, Tilburg University, September 2004.

[7] R. L. Graham, "Bounds on Multiprocessor Timing Anomalies," *SIAM Journal of Applied Mathematics*, vol. 17, pp. 416–429, March 1969.

[8] R. W. Floyd, "Algorithm 97: Shortest Path," *Communications of the ACM*, vol. 5, p. 345, June 1962.