Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Operations Research 33 (2006) 3403–3422

computers & operations research

www.elsevier.com/locate/cor

A tabu search algorithm for the min–max k-Chinese postman problem

Dino Ahr, Gerhard Reinelt*

Institute of Computer Science, University of Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany

Available online 7 December 2005

Abstract

In this paper we present a tabu search algorithm for the min–max k-Chinese postman problem (MM k-CPP). Given an undirected edge-weighted graph and a distinguished depot node, the MM k-CPP consists of finding $k > 1$ tours (starting and ending at the depot node) such that each edge is traversed by at least one tour and the length of the longest tour is minimized. A special emphasis is put on investigating the trade-off between running time effort and solution quality when applying different improvement procedures in the course of the neighborhood construction. Furthermore, different neighborhoods are analyzed. Extensive computational results show that the tabu search algorithm outperforms all known heuristics and improvement procedures.

Scope and purpose Given a road network, the Chinese postman problem (CPP) is to find the shortest postman tour covering all the roads in the network. Applications of the CPP include road maintenance, garbage collection, mail delivery, etc. Since usually large road networks have to be serviced the work load must be distributed among $k \geqslant 2$ vehicles. In contrast to the usual objective to minimize the total distance traveled by the k vehicles (k-CPP), for the min–max k-Chinese postman problem (MM k-CPP) the aim is to minimize the length of the longest of the k tours. This kind of objective is preferable when customers have to be served as early as possible. Furthermore, tours will be enforced to be more balanced resulting in a fair scheduling of tours. Although the CPP and the k-CPP are polynomially solvable, the MM k-CPP is $\mathcal{NP}$-hard. Hence, we must rely on heuristics producing approximate solutions. The purpose of this paper is to present a tabu search algorithm for the MM k-CPP which outperforms all known heuristics. In many cases we obtained solutions which could be proved to be near-optimal or even optimal.
© 2005 Published by Elsevier Ltd.

Keywords: Arc routing; Chinese postman problem; Tabu search; Min–max optimization

* Corresponding author.
E-mail address: gerhard.reinelt@informatik.uni-heidelberg.de (G. Reinelt).

0305-0548/$ - see front matter © 2005 Published by Elsevier Ltd.
doi:10.1016/j.cor.2005.02.011

1. Introduction and related work

1.1. Problem definition and contributions

For the *min–max k-Chinese postman problem* (*MM k-CPP*) we are given an undirected graph $G = (V, E)$, weights $w : E \to \mathbb{R}^+$ for each edge (which we usually interpret as distances), a distinguished depot node $v_1 \in V$ and a fixed number $k > 1$ of postmen. The aim is to find k closed walks (tours) where each tour starts and ends at the depot node and each edge $e \in E$ is covered by at least one tour. In contrast to the usual objective, i.e., minimizing the total distance traveled by the k postmen, for the MM k-CPP we want to minimize the length of the longest of the k tours. This kind of objective function is preferable when the aim is to serve each customer as early as possible. Furthermore, tours will be enforced to be more balanced resulting in a "fair" scheduling of tours.

The MM k-CPP was introduced in [1] and shown to be $\mathcal{NP}$-hard by a reduction from the k-partition problem. Heuristic algorithms for the MM k-CPP were developed in [1,2]. Implementations of these algorithms and computational results were reported in [2].

In this paper we present a tabu search approach for the MM k-CPP. It is based on a set of basic procedures featuring modification and improvement of single tours. Furthermore, different neighborhoods are considered. Initial solutions are taken from the heuristics implemented in the scope of [2].

We ran the tabu search algorithm on a variety of test instances from the literature as well as on random instances. We were able to improve the results obtained by the heuristics in [2] for almost all instances, sometimes considerably. The solution quality is assessed by lower bounds. Furthermore, we compared the behavior of the different neighborhoods and investigated the effect of the improvement procedures on the solution quality and computing time.

The paper is organized as follows. In the remainder of this section we will discuss related work, i.e., existing tabu search approaches for other node and routing problems, and introduce required notation. In Section 2 we develop a set of procedures performing operations on tours which are used in the tabu search algorithm. In Section 3 the tabu search algorithm and the different neighborhoods are described in detail. In the subsequent section we report computational results which show that the tabu search algorithm achieves considerable improvements. Finally, in Section 5, we give conclusions and directions for future work.

1.2. Related work

Tabu search is a meta-heuristic approach for solving combinatorial optimization problems. Basically, tabu search follows a local search scheme, i.e., it starts with a feasible solution, explores the neighborhood of this solution, moves towards a neighborhood solution according to different criteria and proceeds until a fixed number of iterations has been performed. The specific feature of tabu search (compared to local search) is the use of a memory in order to guide the search process in an intelligent way. One aspect of the memory concerns the recency of a move between two adjacent solutions. In order to avoid cycling or sticking in one region of the solution space moves are declared *tabu* for a specified number of iterations thereby giving this method its name. The modern form of tabu search derives from Glover [3,4]. A good overview is given in [5–7].

In recent years tabu search has been applied successfully to many combinatorial optimization problems [8,9]. In order to embed our work in the scientific context we will give an overview of the line of research concerned with tabu search algorithms for routing problems with multiple vehicles. We will roughly distinguish between *node routing* and *arc routing* problems.

1.2.1. Node routing

For the *capacitated vehicle routing problem* (*CVRP*) we are given an undirected graph $G = (V, E)$, edge weights $w(v_i, v_j) \in \mathbb{R}^+$ for all $v_i, v_j \in V$, node demands $d(v) \in \mathbb{R}_0^+$ for all $v \in V$, a distinguished depot node $v_1 \in V$ and a vehicle capacity Q. The aim is to determine a set of cycles (also called *tours*) with minimum sum of edge weights, such that each cycle contains the depot node, each node $v \in V\backslash\{v_1\}$ is traversed exactly once by a cycle and the sum of demands of nodes traversed by each cycle does not exceed the vehicle capacity. The CVRP can be considered as the main target and testbed problem for developing and improving tabu search approaches in the area of routing problems. We briefly review the most important results for the CVRP chronologically. A more comprehensive overview can also be found in [10].

An early approach is due to Willard [11]. A given solution is transformed into a single tour by duplicating the depot node. Neighborhood solutions are defined by 2- or 3-opt [12] exchanges of the current solution. The next current solution is chosen to be the best non-tabu neighborhood solution.

In contrast to the former approach the neighborhood solutions of the algorithm of Pureza and França [13] are obtained by *moving* nodes between tours while preserving feasibility. The best non-tabu move is selected at each iteration.

Concerning the competitiveness with known algorithms at this time the former two approaches do not perform very well. The more sophisticated neighborhood determination in the algorithm of Osman [14] led to more successful results. A neighborhood solution is determined by selecting two tours and performing exchange or shifting of at least λ nodes, $\lambda \in \{0, 1, 2\}$, (a so-called λ*-interchange*). Since computing the whole neighborhood and choosing the best admissible solution can be very time consuming, a faster strategy, choosing the first best admissible solution, has been used in addition.

A rather involved approach, called *Taburoute*, comes from Gendreau et al. [15]. They applied several new features to improve their algorithm. At first the neighborhood solution is again constructed by removing a node from one tour and inserting it into another, but the inserting procedure is guided by a sophisticated scheme (called *GENI*) originally developed for the *traveling salesman problem* (*TSP*) by the same authors [16]. A broader spectrum of neighborhood solutions is obtained by allowing infeasible solutions (according to capacity or maximum route length restrictions) and using an extended objective function which penalizes infeasibility in an appropriate way. Furthermore, nodes which have been moved frequently between routes are also penalized according to the frequency to achieve a better diversification. A randomly chosen tabu tenure is used for each move. Periodically during the search process, the route in which a node has just been inserted is re-optimized by a so-called *unstringing* and *stringing* method [16]. Finally, Taburoute starts with several initial solutions and performs a restricted search on them. Then the main tabu search is started with the most promising start solution found.

The algorithm of Taillard [17] uses ideas from the former two approaches, namely the random tabu tenure and diversification strategy from [15] and the λ-interchange for constructing neighborhood solutions from [14]. Individual routes are re-optimized using an algorithm of Volgenant and Jonker [18]. In order to facilitate a parallel implementation, the main problem is decomposed into subproblems. Periodic exchanges of nodes between adjacent subproblems are performed.

Tabu search approaches considered up to now apply rather classical concepts. The following two algorithms distinguish themselves from the former by including new innovative strategies.

The first new concept, called *adaptive memory*, was introduced by Rochat and Taillard [19] and successfully applied to the CVRP. The idea is to maintain a pool of "good" solutions that is dynamically updated throughout the search process. Parts of the solutions contained in the pool are combined in different ways to form new good solutions. This step is usually performed when there is no progress in the search process.

A further new concept, called *granular tabu search* is due to Toth and Vigo [20]. Here the focus is on restricting the search space in order to speed up computation time. This is accomplished by excluding "long edges" from considerations since they are not expected to be part of an optimal solution. An implementation of this idea based on top of several features of the Taburoute algorithm yielded excellent solutions within short computing times.

There have been also tabu search approaches for the CVRP with min–max objective (*MM CVRP*), i.e., the aim is to minimize the length of the longest cycle. In [21] França et al. describe a tabu search algorithm for the *min–max m-TSP* using insertion and post-optimization procedures from [16]. The *m-traveling salesman problem* (*m-TSP*) is a special case of the CVRP providing unlimited capacity for each vehicle, i.e., $Q = \infty$.

Furthermore, Golden et al. [22] devised tabu search procedures applying adaptive memory for the MM CVRP as well as for the same problem allowing multiple use of vehicles (abbreviated *MM CVRPM*). The approaches for both problems are essentially based on the algorithm of Taillard [17]. For the MM CVRP the min–max objective is enforced by integrating the length of the longest cycle as a distance constraint. In an initial step a pool of solutions consisting of $s \gg m$ cycles is computed (where m is the number of available vehicles). Then these solutions are broken down to solutions having m tours by distributing them according to a load balancing scheme.

1.2.2. Arc routing

For problems from the area of arc routing substantially less research concerning local search and tabu search has been conducted. The main reason is probably that it is much more difficult to define neighborhood solutions in this context.

The counterpart of the CVRP in the area of arc routing is the *capacitated arc routing problem* (*CARP*). We are given an undirected graph $G = (V, E)$, edge weights $w(e) \in \mathbb{R}^+$ for all $e \in E$, edge demands $d(e) \in \mathbb{R}_0^+$ for all $e \in E$, a distinguished depot node v_1 and a vehicle capacity Q. The aim is to determine a set of closed walks with minimum sum of edge weights, such that each closed walk contains the depot node, each edge e with positive edge demand is traversed at least once by a closed walk and the sum of demands of edges traversed by each closed walk does not exceed the vehicle capacity. A special case of the CARP is the *capacitated Chinese postman problem* (*CCPP*) where all edge demands are positive.

Early tabu search resp. simulated annealing approaches for a road gritting problem and a multi-depot gritting problem with several side constraints are due to Li [23] and Eglese [24].

Greistorfer [25] developed a tabu search procedure for the CCPP. He used one kind of neighborhood structure and performs the selection of the neighborhood solution to proceed with according to best improvement and first improvement rules.

The first of a series of successful, recently published meta-heuristics for the CARP is the tabu search algorithm *Carpet* which was developed by Hertz et al. [26]. The main ideas from the Taburoute algorithm [15] for the CVRP have been transferred into the CARP context. Furthermore, some improvement procedures originally developed for the rural postman problem [27] are integrated into this approach.

Computational results demonstrated a clear dominance of Carpet over all known heuristics for the CARP at that time.

Hertz and Mittaz [28] devised a variable neighborhood descent (VND) algorithm based on the idea of variable neighborhood search (VNS) [29]. The VND approach extends the descent local search method by considering different neighborhood structures in each iteration. The neighborhood structures used in [28] operate on up to K routes, where K is the number of available vehicles. For constructing new neighborhood solutions the same basic procedures as used for the Carpet algorithm [26] are utilized. Computational results showed that the VND approach is competitive and for large instance even superior (in terms of solution quality and speed) to Carpet.

In [30,31] Lacomme et al. present the first genetic algorithms for the CARP which are also capable of tackling extensions. At present these line of algorithms are the most successful and outperform all other heuristics for the CARP.

A recent hybrid approach for the CCPP featuring a so-called *scatter search* comes from Greistorfer [32]. It is reported that this approach is competitive to Carpet.

In [33] Beullens et al. come up with a new algorithm which combines the idea of their k-opt approach [34] (originally developed for the *general routing problem* (*GRP*)) with a guided local search scheme (GLS). The key feature of GLS is the usage of a penalizing mechanism in order to direct the search into promising regions of the solution space. Computational experiments demonstrated that the approach is as good as Carpet and could even improve some upper bounds obtained in [26]. Unfortunately, no comparisons to the genetic algorithms of Lacomme et al. [30,31] have been performed.

1.3. Terminology

For the MM k-CPP we have the following input data: a connected undirected graph $G=(V, E)$, weights for each edge $w : E \to \mathbb{R}^+$, a distinguished depot node $v_1 \in V$ and a fixed number $k > 1$ of postmen. A feasible solution, called *k-postman tour*, is a set $\mathscr{C}$ of k closed walks, $\mathscr{C} = \{C_1, \dots, C_k\}$, such that each tour C_i contains the depot node v_1, all edges $e \in E$ are covered by at least one tour C_i and each postman is involved.

For an edge set or a walk F let $|F|$ denote the cardinality of the edge set and the number of edges contained in the walk, respectively.

We extend the weight function w to walks $F = (e_1, \dots, e_p)$ by defining $w(F) = \sum_{i=1}^{p} w(e_i)$. Now, for a k-postman tour $\mathscr{C}$, let us denote the maximum weight attained by a single tour C_i as $w_{\max}(\mathscr{C})$, i.e.,

$$w_{\max}(\mathscr{C}) = \max_{i=1,\dots,k} w(C_i).$$

The objective of the MM k-CPP is to find a k-postman tour $\mathscr{C}^*$ which minimizes $w_{\max}$ among all feasible k-postman tours, i.e.,

$$w_{\max}(\mathscr{C}^*) = \min\{w_{\max}(\mathscr{C}) \mid \mathscr{C} \text{ is a } k\text{-postman tour}\}.$$

Fig. 2 shows a 2-postman $\mathscr{C} = \{C_1, C_2\}$ tour on an example graph G depicted in Fig. 1. The weights of the tours are $w(C_1) = 107$, $w(C_2) = 78$ and hence $w_{\max}(\mathscr{C}) = 107$. Edges of C_1 have dashed line style, edges of C_2 have dotted line style. The example graph G will be used throughout the paper for illustrating algorithms. Note that the weight function does not fulfill the triangle inequality, i.e., $w(\{u, v\}) + w(\{v, w\}) \geqslant w(\{u, w\})$ for $\{u, v\}, \{v, w\}, \{u, w\} \in E$ does not hold in general.

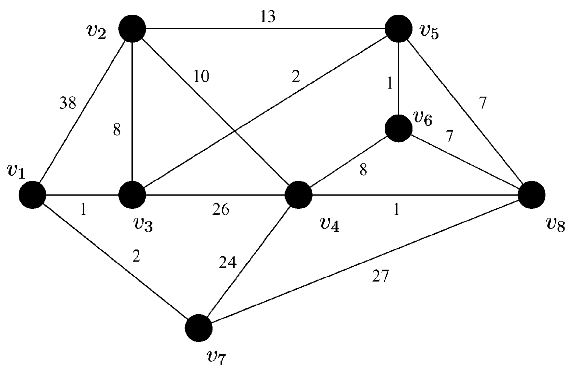

Fig. 1. The example graph G.

We denote by $SP(v_i, v_j)$ the set of edges on the shortest path between nodes $v_i, v_j \in V$. The distance of the shortest path between v_i and v_j is given by $w(SP(v_i, v_j))$. We will use Dijkstra's algorithm [35] and the Floyd–Warshall [36] algorithm to compute single-pair and all pairs shortest paths, respectively, in our implementation. Usually the shortest path computation is performed only once and results are then provided as parameter to the single procedures.

For an edge set $F \subseteq E$ we denote by $V(F) \subseteq V$ the set of nodes being incident to some edge from F. For a node $v \in V$ we denote by $\delta(v)$ the set of edges being incident to v.

2. Basic procedures

In this section we want to develop a set of basic procedures which allow us to easily perform modifications on feasible solutions for the MM k-CPP. These procedures will be used to construct neighborhood solutions in Section 3. Furthermore, we will present three different improvement procedures which can be applied after each modification.

2.1. Merging and separating edges

The main operations needed for modifying single tours are addition and removal of edges. Of course we have to take care that when modifying a tour C_i by adding or removing edges this tour remains a closed walk. Furthermore, when removing edges we must ensure that the depot node is still contained in C_i afterward. This additional work is performed in the procedures *MergeWalkWithTour* and *SeparateWalkFromTour*. In the scope of our tabu search algorithm we only encounter the situation that single edges or edge sets representing walks have to be merged or separated.

Let us now consider the procedure *MergeWalkWithTour* in detail.

Algorithm: MERGEWALKWITHTOUR
Input: The tour C_i and the walk H to be merged.
Output: The tour $\tilde{C}_i$ which represents a feasible tour formed from edges contained in C_i and H and possibly additional edges.

(1) Remove those edges from the beginning and the end of H which also occur in C_i and let $\hat{H}$ be the result.
(2) Let u and v be the endnodes of $\hat{H}$. Determine the node t on C_i which minimizes $w(SP(u,t)) + w(SP(v,t))$.
(3) Let $\tilde{C}_i$ evolve from splicing $SP(t,u), \hat{H}, SP(v,t)$ into C_i at node t.

Step (1) costs $O(|H| + |C_i|)$. Step (2) needs $O(|C_i|)$ (the shortest path information is given as input parameter). Step (3) requires traversing the walk $SP(t,u), H, SP(v,t)$ which needs at most $O(|E|)$. Hence, the overall required time is $O(|E|)$.

The procedure *SeparateWalkFromTour* is rather straightforward except that we have to take care that the depot node remains in the tour.

Algorithm: SeparateWalkFromTour

Input: The tour C_i and the walk H to be separated.

Output: The tour $\tilde{C}_i$ which represents a feasible tour formed from edges of C_i remaining after removing edges from H and possibly additional edges needed to re-establish feasibility.

(1) Let u and v be the endnodes of H ($u = v$ is possible). Check whether the depot node v_1 is an internal node of H.
(2) Remove all edges of H from C_i and let $\hat{C}_i$ be the result. Check whether the depot node v_1 is contained in $\hat{C}_i$.
(3) We have to consider two cases when connecting u and v:
    - If v_1 is an internal node of H and v_1 is not contained in $\hat{C}_i$ then let $\tilde{C}_i$ evolve from $\hat{C}_i$ by connecting u and v with $SP(u, v_1), SP(v_1, v)$.
    - Otherwise let $\tilde{C}_i$ evolve from $\hat{C}_i$ by connecting u and v with $SP(u,v)$.

Step (1) costs $O(|H|)$. Step (2) needs $O(|C_i|)$. Step (3) needs at most $O(|E|)$. Hence the overall time is $O(|E|)$.

Note, that when applying the procedures *MergeWalkWithTour* and *Separate WalkFromTour* one has to pay attention that the overall k-postman tour remains feasible. We will care about this issue when creating neighborhood solutions in Section 3.2.

Let us illustrate the above procedures by performing modifications of the 2-postman tour depicted in Fig. 2. Fig. 3 shows the 2-postman tour after applying the operation *SeparateWalkFromTour* on C_1 (dashed edges) with $H = \{v_4, v_6\}, \{v_6, v_8\}$. The endnodes v_4 and v_8 have been connected by the shortest path edge $\{v_4, v_8\}$. The weight of the tour C_1 has decreased by 14 and now amounts to $w(C_1) = 93$, but the current 2-postman tour is not feasible since edge $\{v_6, v_8\}$ is not serviced. Fig. 4 shows the tour depicted in Fig. 3 after applying the operation *MergeWalkWithTour* on C_2 (dotted edges) with $H = \{v_4, v_6\}, \{v_6, v_8\}$. For example, the node $t = v_4$ minimizes the sum of shortest path distances to the endnodes v_4 and v_8. Hence again edge $\{v_4, v_8\}$ has been added to C_2. Though the weight of tour C_2 has increased by 16, we have improved the objective function value by 13 because now we have $w(C_1) = 93$ and $w(C_2) = 94$ and hence $w_{\max}(\mathscr{C}) = 94$.

2.2. Improving tours

Solutions obtained by heuristics and in particular by modification of single tours often leave potential for further improvement. Such *improvement procedures* have been proven highly successful in

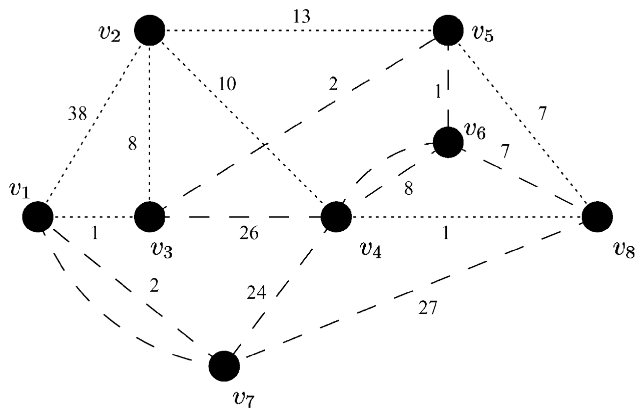

Fig. 2. A 2-postman tour $\mathscr{C} = \{C_1, C_2\}$ on G. Edges of C_1 have dashed line style, edges of C_2 have dotted line style.

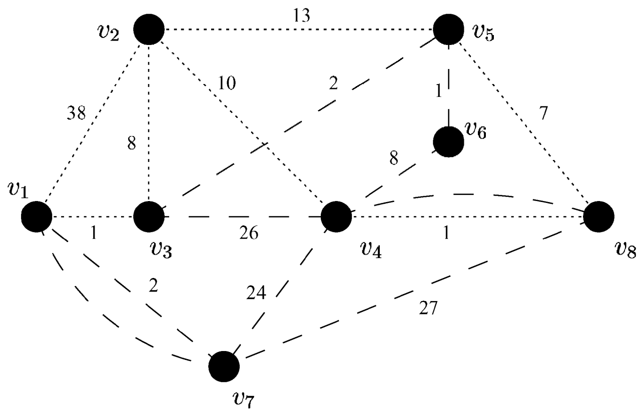

Fig. 3. The 2-postman tour shown in Fig. 2 after applying the operation *SeparateWalkFromTour* on C_1 (dashed edges) with $H = \{v_4, v_6\}, \{v_6, v_8\}$.

producing better solutions. Hence it seems natural to apply improvement procedures after each modification of a single tour by *MergeWalkWithTour* and *SeparateWalkFromTour*. However, we have to be aware that, when computing neighborhood solutions, many modification steps will be performed and thus the improvement procedure will be executed quite often. Therefore the overall computing time for the tabu search algorithm will depend heavily on the effort spent in the improvement procedure. In order to investigate the trade-off between effort and solution quality we developed three improvement procedures of different complexity. But before presenting these procedures we have to introduce the notion of *required edges* edges which is essential for developing improvement procedures in our context.

The main difficulty in modifying the single tours C_i of a k-postman tour $\mathscr{C}$ lies in maintaining feasibility of $\mathscr{C}$. Feasibility means that all tours remain closed walks containing the depot node and that each edge is traversed by at least one tour. For example, if we want to remove one or several edges from a tour C_i we have to ensure that C_i remains a closed walk (by eventually adding new edges) and moreover that those edges which have been deleted from C_i are traversed by some other tours. This task is much more

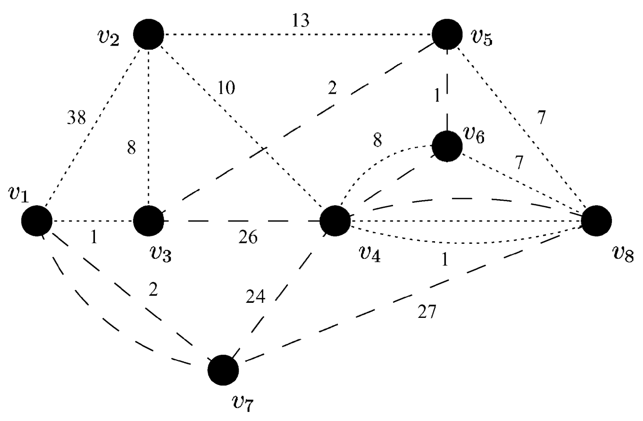

Fig. 4. The 2-postman tour shown in Fig. 3 after applying the operation *MergeWalkWithTour* on C_2 (dotted edges) with $H = \{v_4, v_6\}, \ \{v_6, v_8\}$.

intricate than modification of tours in the node routing context since for node routing problems we do not have to care about which edges are used to connect nodes (apart from favoring cheap edges).

Distinguishing between *required edges* and *non-required edges* helps in performing admissible modifications. For example, for the CARP this information is given explicitly, namely those edges with positive demand are required. For the MM k-CPP we do not have explicit information since all edges have to be served. Nevertheless, it is possible to classify edges to be required or not in our context. Given a k-postman tour $\mathscr{C}$ let us consider a single tour C_i. We say that an edge e, which is traversed by C_i but not traversed by any other tour C_j, $j \neq i$, is *required for* C_i. In order to make this classification we need to know how many times an edge is contained in $\mathscr{C}$ and in single tours C_i, respectively.

More formally, let us denote by *edge frequencies* the information how often edges occur and let $\phi_i(e)$ denote the *frequency of e occurring in* C_i and $\phi(e) = \sum_{i=1}^{k} \phi_i(e)$ the *frequency of e occurring in all tours*. An edge e is called *redundant for* C_i if $\phi_i(e) \geqslant 1$ and $\phi(e) > \phi_i(e)$. On the contrary, an edge e is called *required for* C_i if $\phi_i(e) \geqslant 1$ and $\phi(e) = \phi_i(e)$. The time complexity for computing edge frequencies is $O(k|E|)$.

Now it is easy to state what exactly we expect from an improvement procedure, namely given a single tour C_i from a k-postman tour $\mathscr{C}$ try to decrease the tour length while keeping the required edges in C_i. But this is exactly the *rural postman problem* (*RPP*) which is $\mathscr{NP}$-hard [37]. Since we want to apply our improvement procedures as a post-processing step for improving a neighborhood solution it is clear that we will focus on fast heuristic procedures.

Let us now turn to the first improvement procedure called *RemoveReplicateEdgesKeepingParity* which tries to remove superfluous edges in a given tour C_i. It is based on the simple observation that for edges e occurring $n \geqslant 3$ times in C_i we can remove $n-2$ times ($n-1$ times) e if n is even (odd) because we keep the parity of the incident nodes and also the connectivity of the tour. This procedure can be accomplished in $O(|C_i|)$, however, in general the resulting set of edges $\tilde{C}_i$ does not represent a closed walk anymore. Since reestablishing a closed walk can be accomplished in linear time too (which will be explained next), *RemoveReplicateEdgesKeepingParity* represents a simple and fast heuristic.

The procedure *ReorderToClosedWalk* reorders the edges of a given edge set C_i to obtain a closed walk. Necessary and sufficient conditions for being able to do this are that all nodes from $V(C_i)$ have even

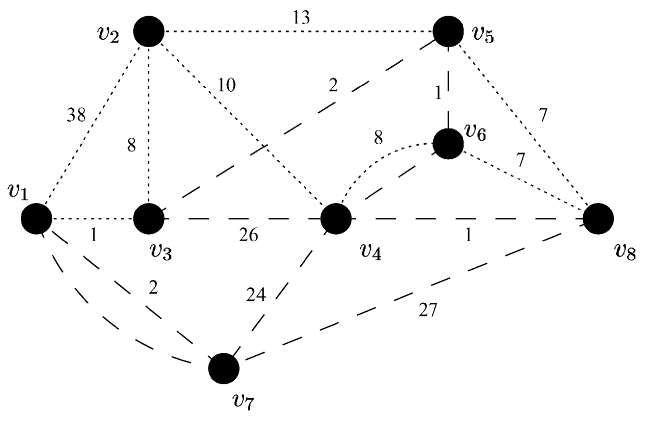

Fig. 5. The 2-postman tour shown in Fig. 4 after applying the operation *RemoveEvenRedundantEdges* on C_2 (dotted edges).

degree (according to C_i) and the graph $G[C_i]$ induced by C_i is connected. The algorithm is based on the well-known end-pairing algorithm of Edmonds and Johnson [38] and basically constructs a series of closed walks which are successively spliced together.

The second improvement procedure, called *RemoveEvenRedundantEdges*, basically works exactly like *RemoveReplicateEdgesKeepingParity* but it does a little bit more. In particular when considering a tour C_i and a redundant edge e, i.e., $\phi(e) > \phi_i(e)$, with even frequency, i.e., $\phi_i(e) \equiv 0 \bmod 2$, e will be removed completely from C_i if the remaining C_i remains connected and still contains the depot node. The time complexity is $O(|C_i|^2)$ because for each edge e from C_i we have to check whether C_i without e remains connected and still contains the depot node which can be done in $O(|C_i|)$ for each. Again *ReorderToClosedWalk* has to be applied afterward.

Fig. 5 shows the 2-postman tour depicted in Fig. 4 after applying the operation *RemoveEvenRedundantEdges* on C_2 (dotted edges). The edge $\{v_4, v_8\}$ is even and redundant for C_2 and can be removed since C_2 remains connected. Now $w(C_2) = 92$ and the objective function value decreased by 1 because now $w_{\max}(\mathscr{C}) = 93$.

The last improvement procedure is based on the routine *Shorten* which was developed as an improvement procedure for the RPP [27] and used for the tabu search algorithm *Carpet* for the CARP [26]. The basic idea is to identify redundant walk segments P in the tour C_i and replace them by a shortest path connecting the end nodes of P. We will call this procedure *ShortenRequiredEdgeConnections*.

```
Algorithm: SHORTENREQUIREDEDGECONNECTIONS
Input: A single tour $C_i$ from a $k$-postman tour $\mathscr{C}$.
Output: A possibly improved tour $\tilde{C}_i$.

(1) Consider each possible starting edge and each of the two orientations of the tour $C_i$.
    (1.1) Compute edge frequencies.
    (1.2) Traverse $C_i$.
        (1.2.1) Construct the longest possible walk $P$ (while traversing $C_i$) consisting of redundant edges.
                Let $v_h$ be the origin and $v_j$ be the terminus of walk $P$.
```

(1.2.2) Try to enlarge P.
Traverse the tour C_i further on—building up walk Q—until a redundant edge (v_l, v_j) entering v_j is found. If such a walk $Q = \{v_j, \ldots, v_l, v_j\}$ is found, reverse the orientation of Q in C_i and continue with step (1.2.1) starting again at v_h.

(1.2.3) Replace P by the shortest path $SP(v_h, v_j)$ if $w(SP(v_h, v_j)) < w(P)$. Continue with step (1.2) with the edge following the last edge of P.

(2) If the depot node v_1 is not contained in C_i anymore, choose the node v_t on C_i which has minimum distance to v_1 and merge two times $SP(v_1, v_t)$ with C_i in order to connect the depot node with a closed walk to C_i.

In step (1.1) we always have to recompute the edge frequencies because we modify them in step (1.2.1) to keep book of redundant and required edges. The loop of step (1) is executed $O(2|C_i|)$ times. Step (1.1) costs $O(k|E|)$. Step (1.2) is difficult to analyze. In the worst case it could happen that for each edge added to path P a required edge would follow but we could find an appropriate walk Q and then we would start again at the beginning of P. Hence step (1.2) has worst case time of $O(|C_i|^2)$. Step (2) needs only $O(|C_i|)$. Hence the worst case running time is $O(|C_i|^3)$.

We refer the reader to [26,27] for illustrating examples of the procedure *ShortenRequiredEdge-Connections*.

3. The tabu search algorithm

We will first explain the generic tabu search procedure. Then we will present three different neighborhood structures.

3.1. The generic tabu search algorithm

The generic tabu search algorithm for the MM k-CPP works as follows. In each iteration we consider a so-called *currentSolution* (which is initialized with an input solution $\mathscr{C}$) and compute a set of neighborhood solutions N(*currentSolution*). A neighborhood solution is obtained by performing slight modifications on the current solution (neighborhoods will be presented in the next section). When going from one solution $\mathscr{C}_i$ to a neighborhood solution $\mathscr{C}_j$ (which will be called *move*) we want to maximize the improvement of the objective function value $w_{\max}(\mathscr{C}_i) - w_{\max}(\mathscr{C}_j)$ (which will be called *move value*). Hence the neighborhood solutions will be considered in decreasing order of their move value and the non-tabu neighbor solution with the best move value is chosen to be the next current solution. In the case that the neighborhood solution is tabu but better than the current best solution the tabu rule will be ignored. A solution is declared tabu according to a recency criterion, namely in the case when the move from the current solution to the neighbor solution has already appeared in the *tabuTenure* last iterations (the exact tabu criterion depends on the neighborhood and will be explained in the next section). The algorithm terminates if no improvement of the best solution has been achieved in the last *maxNOfItsWithoutImprovement* iterations. More formally the algorithm works as follows:

Algorithm: MMKCPPGENERICTABUSEARCH

Input: A k-postman tour $\mathscr{C}$, the maximum number of iterations without improvement *maxNOfItsWithoutImprovement*, a tabu tenure *tabuTenure* and a flag *improvementProcedure*.

```
Output: A possibly improved k-postman tour $\tilde{\mathscr{C}}$.
(1) Initialize.
    • $bestSolution = currentSolution = \mathscr{C}$
    • $bestSolutionValue = currentSolutionValue = w_{\max}(\mathscr{C})$
    • $nOfItsWithoutImprovement = 0$
(2) while $nOfItsWithoutImprovement < maxNOfItsWithoutImprovement$.
    (2.1) Increment nOfItsWithoutImprovement.
    (2.2) Compute a list of neighborhood solutions N(currentSolution) (with parameter improvement-
          Procedure) in decreasing order of their move value.
    (2.3) Let neighborSolution be the first solution of the list which is either non-tabu or tabu but
          $neighborSolutionValue < bestSolutionValue$ (if no such solution exists the algorithm termi-
          nates).
          Set $currentSolution = neighborSolution$ and $currentSolutionValue = neighborSolutionValue$.
          If $currentSolutionValue < bestSolutionValue$ then $bestSolution=currentSolution$, bestSolution-
          $Value = currentSolutionValue$ and $nOfItsWithoutImprovement = 0$.
```

3.2. Neighborhoods

Given a current solution $\mathscr{C}$ there is a huge number of possibilities to construct neighborhood solutions. It is clear that one has to consider specialized and promising neighborhoods with restricted size. As a first restriction we will confine ourselves to neighborhood solutions where only two tours, namely the longest tour C_i and any other tour $C_j,\ j \neq i$, are modified. The three neighborhoods presented in the following mainly differ in the way they exchange edges between C_i and C_j.

In the following when talking about *merging* and *separating* edges or walks we refer to the application of the procedures *MergeWalkWithTour* and *SeparateWalkFromTour*, respectively, introduced in Section 2.1. As already mentioned in Section 2.2 we will apply improvement procedures after merging and separating edges or walks.

The *TwoEdgeExchange* neighborhood successively considers two consecutive edges e and f in the longest tour C_i. These edges are separated from C_i and merged with any other tour $C_j,\ j \neq i$. Hence we obtain $(k-1)(|C_i|-1)$ neighborhood solutions. After separating e and f the selected improvement procedure is applied to C_i and after merging e and f with $C_j,\ j \neq i$, the improvement procedure is applied to C_j. For the tabu criterion, two moves from $\mathscr{C}$ to $\mathscr{C}_1$ resp. $\mathscr{C}_2$ obtained by separating e_1, f_1 resp. e_2, f_2 from C_{i_1} resp. C_{i_2} and merging them with C_{j_1} resp. C_{j_2}, are considered to be identical if $i_1 = i_2$, $e_1 = e_2$, $f_1 = f_2$ and $j_1 = j_2$.

An example for the construction of a *TwoEdgeExchange* neighborhood solution is also contained in the examples illustrating the operations *SeparateWalkFromTour* and *MergeWalkWithTour* (Figs. 2–4). If we consider the solution depicted in Fig. 2 to be the current solution $\mathscr{C}$, a possible neighborhood solution $\tilde{\mathscr{C}}$ (for $C_i = C_1$, $C_j = C_2$, $e = \{v_4, v_6\}$, $f = \{v_6, v_8\}$) is given in Fig. 4.

The *RequiredEdgeWalkExchange* neighborhood successively considers walks $H = P_1, e, P_2$ in the longest tour C_i where e is required for C_i and P_1 and P_2 contain as many redundant edges for C_i—preceding and following e, respectively—as possible. The walk H is separated from C_i and e is merged with any other tour $C_j,\ j \neq i$. Hence we obtain at most $(k-1)|C_i|$ neighborhood solutions in the case that all edges are required. Usually there will be much less neighborhood solutions. After separating H, the selected improvement procedure is applied to C_i and after merging e with C_j, $j \neq i$, the improvement procedure is applied to C_j. Greistorfer [25] used a similar neighborhood structure for the CCPP. For the tabu criterion, two moves from $\mathscr{C}$ to $\mathscr{C}_1$ resp. $\mathscr{C}_2$ obtained by separating $H_1 = P_1^1, e_1, P_2^1$ resp. $H_2 = P_1^2, e_2, P_2^2$ from C_{i_1} resp. C_{i_2} and merging e_1 resp. e_2 with C_{j_1} resp. C_{j_2}, are considered to be identical if $i_1 = i_2$, $e_1 = e_2$ and $j_1 = j_2$.

The *SingleRequiredEdgeExchange* neighborhood successively considers required edges e in the longest tour C_i. The edge e is separated from C_i and merged with any other tour C_j, $j \neq i$. Hence we obtain at most $(k-1)|C_i|$ neighborhood solutions. After separating e from C_i and merging e with C_j, $j \neq i$, the improvement procedure is applied to C_i and afterward to C_j. Note, that without application of improvement procedures the longest tour C_i will not be improved for instances having all weights fulfilling the triangle inequality, since e will be reinserted as shortest connection between its endnodes by the routine *SeparateWalkFromTour*. For the tabu criterion, two moves from $\mathscr{C}$ to $\mathscr{C}_1$ resp. $\mathscr{C}_2$ obtained by merging e_1 resp. e_2 from C_{i_1} resp. C_{i_2} with C_{j_1} resp. C_{j_2}, are considered to be identical if $i_1 = i_2$, $e_1 = e_2$ and $j_1 = j_2$.

The above neighborhood in connection with the improvement procedure *ShortenRequiredEdgeConnections* was used in the *Carpet* tabu search algorithm for the CARP [26].

4. Computational results

For our computational experiments we used instances of related problems from the literature (after appropriate adaptions) as well as a set of 5 randomly generated instances which are partially also used in [2]. In particular the instance set from the literature consists of 24 RPP instances from [39], 2 RPP instances from [40], 8 GRP instances from [41], 10 CARP instances from [42], 23 CARP instances from [43] and 2 CARP instances from [44]. The largest graphs have sizes of $|V| = 200$ and $|E| = 392$. We used the heuristics presented in [2] to produce initial solutions to improve on. Furthermore, we always considered the range $k = 2, \ldots, 10$ for the number of postmen (except for instance sets which contain very small instances). All computations were performed on a SUN Ultra 80 with Ultra Sparc IIi processors at 450 MHz.

In a preliminary experiment the goal was to find an appropriate setting for the parameter *tabuTenure*. We performed computations (for each combination of initial heuristic, tabu search neighborhood and improvement procedure) on the 5 randomly generated instances while using the values {5, 10, 15, 20} for *tabuTenure*. We imposed a time limit of 600 s on each run. We found, that for some instances the tabu tenure did not have any impact on the solution, but for most instances the values 15 and 20 led to solutions of best quality. Therefore, we decided to fix the parameter *tabuTenure* to 20. The parameter *maxNOfIterationsWithoutImprovement* is set to 100.

In a second experiment we investigated the impact of the three different neighborhood structures on the solution quality. This was accomplished by comparing the results of the three neighborhoods for each of the four configurations of post-processing by improvement procedures (one configuration is without using an improvement procedure). Again we used the 5 randomly generated instances and also the 2 instances from [40]. The time limit was set to 600 s on each run.

Roughly, we observed that the *TwoEdgeExchange* (*TEE*) neighborhood was superior to the *RequiredEdgeWalkExchange* (*REWE*) neighborhood for the cases of applying *RemoveReplicateEdgesKeepingParity*, *RemoveEvenRedundantEdges* and when no improvement procedure was used. The *SingleRequiredEdgeExchange* (*SREE*) neighborhood could not achieve any best solution in these three cases. In the case of using *ShortenRequiredEdgeConnections* the best solutions are almost equally distributed among the three neighborhoods.

We now want to go into more detail and try to give reasons for the behavior we observed in the second experiment. As already mentioned, in general the TEE neighborhood is superior (in the first three cases) but for an increasing number of postmen k we found that solutions obtained for the REWE neighborhood become better and then often outperform the solutions of the TEE neighborhood. This phenomenon can be explained by the following consideration. For k small it is less likely that single tours overlap and hence almost all edges of a single tour are required. For the REWE neighborhood this would result in H (the walk to be exchanged) consisting only of one required edge. An exchange of only one required edge (which is exactly the case of the SREE neighborhood) is not very effective which we have already observed for the SREE neighborhood. But with increasing k it is more likely that single tours serve the same edges and hence the number of redundant edges grows. This, in turn, leads to larger walks H to be exchanged in the REWE neighborhood and hence to probably more promising neighbor solutions. The TEE neighborhood has a fixed exchange scheme and thus is independent of the value of k. This is probably the reason for the TEE neighborhood being superior in most cases. When using *ShortenRequiredEdgeConnections as improvement* procedure the best solutions are almost equally distributed among the three neighborhoods. The reason is that this improvement procedure performs substantial modifications of a solution and it seems to be a matter of accident which solution is appropriate for achieving best improvements. So probably *ShortenRequiredEdgeConnections* can often turn "bad solutions" in better solutions than improving on a fairly good solution.

The next experiment was concerned with investigating the effect of the improvement procedures. Recall that in the tabu search algorithm the selected improvement procedure will be applied each time a neighborhood solution is constructed (cf. Section 3.2). In order to put the running time in relation to the solution quality we imposed two different time limits on each run, namely 60 s for rather short runs and 600 s for rather long runs. Furthermore, runs have been performed for all three neighborhoods. Again we used the 5 randomly generated instances and also the 2 instances from [40]. These seven instances can be roughly divided into three small instances with $|V| \in [20, 40]$ and $|E| \in [32, 70]$ and four larger instances with $|V| \in [90, 102]$ and $|E| \in [144, 199]$.

For all three neighborhoods we observed the same effects of the improvement procedures. Namely, for the short runs the improvement procedure *RemoveEvenRedundantEdges* was superior for all instances except for the smallest instance with $|V| = 20$ and $|E| = 32$. For the long runs the improvement procedure *ShortenRequiredEdgeConnections* was superior for the three small instances but for the large instances again *RemoveEvenRedundantEdges* was better. In only a few cases *RemoveReplicateEdgesKeepingParity* yielded the best result. This experiment clearly shows that the improvement procedure *ShortenRequired-EdgeConnections* (with its high time complexity) is not favorable if we have time restriction or deal with large graphs. The procedure *RemoveEvenRedundantEdges* seems to make an excellent compromise between time complexity and quality and for large instances there is even no alternative in choosing this improvement procedure.

The last experiment compared the results of the tabu search algorithm with the results of existing heuristics (including improvement procedures) and lower bounds from [2]. Computations were performed on the whole instance set. The time limit was set to 600 s on each run. Since listing results for each instance and each number of postmen $k = 2, \dots, 10$ would produce too extensive tables we restricted the information to average results over all postmen. For tables with explicit results please contact the authors.

Table 1
Random generated instances, $k = 2, \ldots, 10$

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
Random1	20	33	3.68	4.61
Random2	20	32	7.15	2.17
Random3	40	70	13.23	8.02
r2d4nb5	100	160	5.65	14.44
r1d5nb5	100	199	5.48	16.90

Table 2
RPP instances from [39], $k = 2, \ldots, 7$

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
P01	11	13	0.00	0.31
P02	14	33	13.21	2.78
P03	28	57	6.73	8.00
P04	17	35	8.59	4.02
P05	20	35	10.36	7.26
P06	24	46	10.26	7.76
P07	23	47	16.76	1.70
P08	17	40	16.07	5.07
P09	14	26	6.73	8.83
P10	12	20	11.55	3.83
P11	9	14	4.46	0.00
P12	7	18	15.16	3.32
P13	7	10	4.41	2.87
P14	28	79	9.15	4.75
P15	26	37	6.16	5.31
P16	31	94	8.11	3.42
P17	19	44	12.08	3.11
P18	23	37	8.39	2.56
P19	33	54	6.23	2.35
P20	50	98	8.14	5.08
P21	49	110	5.72	4.98
P22	50	184	2.30	1.65
P23	50	158	6.18	3.67
P24	41	125	4.47	5.71

Table 1 shows results for the randomly generated instances, Table 2 for instances from [39], Table 3 for instances from [40], Table 4 for instances from [41], Table 5 for instances from [42], Table 6 for instances from [43] and Table 7 for instances from [44].

The first three columns of the tables contain the instance name, the number of nodes $|V|$ and the number of edges $|E|$ of the instance. The last two columns contain the average improvement of the tabu search algorithm over the existing heuristics and the average gap between the results of the tabu search algorithm and the lower bounds, respectively. The average is drawn over the range of postmen (usually

Table 3
RPP instances from [40], $k = 2, \ldots, 10$

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
ALBAIDAA	102	160	4.93	9.64
ALBAIDAB	90	144	5.45	5.85

Table 4
GRP instances from [41], $k = 2, \ldots, 10$

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
ALBA_3_1	116	174	2.18	10.32
MADR_3_1	196	316	1.79	15.32
GTSP1	150	297	2.75	15.99
GTSP2	150	296	2.40	17.44
GTSP3	152	296	1.55	10.96
GTSP4	195	348	1.90	19.69
GTSP5	200	392	2.41	19.85
GTSP6	200	386	2.02	19.55

Table 5
CARP instances from [42], $k = 2, \ldots, 10$

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
1A	24	39	7.72	1.43
2A	24	34	4.22	3.53
3A	24	35	2.75	1.70
4A	41	69	8.82	4.31
5A	34	65	11.06	7.55
6A	31	50	8.36	3.51
7A	40	66	8.74	8.20
8A	30	63	11.32	5.27
9A	50	92	8.32	9.65
10A	50	97	6.41	13.42

$k = 2, \ldots, 10$). In more detail the improvement for a fixed k is determined as follows. Let x be the value of the best heuristic solution (including improvement procedures) from [2] and y be the best solution value obtained by the tabu search algorithm for all possible configurations of neighborhood structures and improvement procedures. Then the improvement is determined as $(x - y) \cdot 100/x$ (note that $x \geqslant y$ holds because the tabu search algorithm improves on the heuristic solutions). Similarly, for the gap let z be the best lower bound obtained by the lower bounds SPT-LB and CPP/k-LB presented in [2]. Then the gap is determined as $(y - z) \cdot 100/y$.

The solutions obtained by the heuristics could be improved by the tabu search algorithm in almost all cases. The improvement is often considerable, in many cases about 10% for some instance/postman configurations even about 20–30%. A different behavior can be observed for the instance sets from [41] (Table 4) and [44] (Table 7) which contain the largest instances of the test set. Here the average

Table 6
CARP instances from [43], $k = 2, \ldots, 9$

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
gdb1	12	22	11.56	1.57
gdb2	12	26	16.51	0.60
gdb3	12	22	12.71	0.60
gdb4	11	19	10.27	1.70
gdb5	13	26	11.09	3.16
gdb6	12	22	10.30	1.70
gdb7	12	22	13.88	3.29
gdb8	27	46	10.80	8.36
gdb9	27	51	14.41	4.05
gdb10	12	25	16.50	7.22
gdb11	22	45	13.53	5.34
gdb12	13	23	9.81	0.80
gdb13	10	28	4.65	0.10
gdb14	7	21	19.85	3.73
gdb15	7	21	14.18	0.00
gdb16	8	28	15.70	1.96
gdb17	8	28	12.92	0.89
gdb18	9	36	14.22	1.61
gdb19	8	11	3.78	1.19
gdb20	11	22	9.19	1.05
gdb21	11	33	13.04	3.37
gdb22	11	44	9.12	0.85
gdb23	11	55	8.18	0.88

Table 7
CARP instances from [44], $k = 2, \ldots, 10$

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
egl-e4-A	77	98	1.34	11.95
egl-s4-A	140	190	2.46	16.23

Table 8
CARP instances from [44], $k = 2, \ldots, 10$, no time limit

Instance	$\|V\|$	$\|E\|$	Average impr. (%)	Average gap (%)
egl-e4-A	77	98	4.23	9.32
egl-s4-A	140	190	7.27	11.88

improvement of 1.9% and 2.1%, respectively, is rather low. This, however, can be explained by the limited amount of time (600 s) which does not allow to create as many neighborhood solutions as required for obtaining better solutions. This is confirmed by the results contained in Table 8 which show an average improvement of 5.75% for the instance set from [44] when the time limit is dropped.

Although in many cases the improvement is about 10% we can also observe many instances/postman configurations with only low improvement "on the first sight". But if we have a closer look we discover that in most of these cases the heuristic solution is already very good, i.e., very close to the best lower bound. It is clear that a large improvement is not possible in those cases. In spite of this, the tabu search algorithm often determines a better solution which is even optimal in many cases.

A final comment concerns the lower bounds in order to explain the sometimes large gaps occurring. The CPP/k-LB is obtained by computing an optimum 1-postman tour (which is the Chinese postman problem and polynomial solvable [38]) and dividing its weight by k. In general it produces good results only for $k = 2, 3$ because the more postmen are in use the more overlappings will happen. However, these overlappings are not taken into account. The SPT-LB is the shortest tour length required to serve the edge farthest away from the depot node. It is independent of the number of postmen k and yields good results only for an appropriate ratio of k and the size (and structure) of the instance. For small and medium sized instances and $k = 8, 9, 10$ we observed that the value of SPT-LB often coincides with the optimum solution value. Hence the sometimes large gap values, in particular for $k = 4, 5, 6, 7$, are rather based on bad lower bounds than on bad solutions obtained by the tabu search algorithm.

5. Conclusion

We have presented a tabu search algorithm for the MM k-CPP. Besides the usual features of a tabu search approach we focus on investigating the trade-off between running time effort and solution quality when applying improvement procedures of different complexity in the course of the neighborhood construction. We devised three different improvement procedures *RemoveReplicateEdgesKeepingParity*, *RemoveEvenRedundantEdges* and *ShortenRequiredEdgeConnections* of linear, quadratic and cubic running time and made the following observations. The improvement procedure *ShortenRequiredEdgeConnections* with its high time complexity yields very good results but is too time consuming for large instances or when time restrictions are imposed. An excellent compromise between running time and solution quality is made by *RemoveEvenRedundantEdges* and for larger instances there is no alternative for an effective improvement procedure. The fast but rather straightforward procedure *RemoveReplicateEdgesKeepingParity* could not compete in most cases.

Furthermore, we analyzed three different neighborhoods *TwoEdgeExchange* (*TEE*), *RequiredEdgeWalkExchange* (*REWE*) and *SingleRequiredEdgeExchange* (*SREE*). We found that when applying the improvement procedure *ShortenRequiredEdgeConnections* the choice of the neighborhood structure did not matter because extensive modifications of intermediate solutions are accomplished. For the remaining two improvement procedures we observed that the TEE neighborhood with its fixed exchange scheme is superior in most cases. However, for an increasing number of postmen k the REWE neighborhood becomes better since the growing number of redundant edges leads to exchanges of larger walks and this, in turn, to promising neighbor solutions.

We have compared the results of the tabu search algorithm with the results of existing heuristics (including improvement procedures) from [2] on a large set of instances from the literature and randomly generated instances. The tabu search algorithm could achieve improvements in almost all cases. The improvement is in many cases about 10% for some instance/postman configurations even about 20–30%. By using lower bounds from [2] we were able to assess the quality of the solutions. In many cases the tabu search algorithm yielded near optimal or even optimal solutions. Our results and our set of benchmark instances can serve as testbed for further research on the MM k-CPP.

Our future work will comprise further fine tuning and enhancements of the tabu search algorithm, e.g. by using a randomized tabu tenure and introducing a frequency criterion. Furthermore, we intend to investigate new neighborhoods, e.g. an *m*-edge exchange neighborhood, and develop new improvement procedures. In order to attack larger instances we plan to parallelize the tabu search algorithm.

References

[1] Frederickson GN, Hecht MS, Kim CE. Approximation algorithms for some routing problems. SIAM Journal on Computing 1978;7(2):178–93.

[2] Ahr D, Reinelt G. New heuristics and lower bounds for the min–max *k*-Chinese postman problem. In: Möring R, Raman R, editors. Algorithms—ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17–21, 2002. Proceedings, Lecture notes in computer science, vol. 2461. Berlin: Springer; 2002. p. 64–74.

[3] Glover F. Heuristic for integer programming using surrogate constraints. Decision Science 1977;8:156–66.

[4] Glover F. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 1986;13:533–49.

[5] Glover F. Tabu search, Part I. ORSA Journal on Computing 1990;1:190–219.

[6] Glover F. Tabu search, Part II. ORSA Journal on Computing 1991;2:4–32.

[7] Glover F, Laguna M. Tabu search. In: Reeves C, editor. Modern heuristic techniques for combinatorial problems. Oxford: Blackwell Scientific Publications; 1993. p. 70–150.

[8] Reeves C, editor. Modern heuristic techniques for combinatorial problems. Oxford: Blackwell Scientific Publications; 1993.

[9] Aarts E, Lenstra JK, editors. Local search in combinatorial optimization. New York: Wiley, 1997.

[10] Gendreau M, Laporte G, Potvin J-Y. Metaheuristics for the capacitated VRP. In: Toth P, Vigo D, editors. The vehicle routing problem, SIAM monographs on discrete mathematics and applications. Philadelphia, PA: SIAM; 2002. p. 129–54 [Chapter 6].

[11] Willard JAG. Vehicle routing using *r*-optimal tabu search. Master’s thesis, The Management School, Imperial College, London, 1989.

[12] Lin S. Computer solutions of the traveling salesman problem. Bell System Technical Journal 1965;44:2245–69.

[13] Pureza VM, França PM. Vehicle routing problems via tabu search metaheuristic. Technical Report CRT-347, Centre for Research on Transportation, Montreal, Canada, 1991.

[14] Osman IH. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 1993;41:421–51.

[15] Gendreau M, Hertz A, Laporte G. A tabu search heuristic for the vehicle routing problem. Management Science 1994;40:1276–90.

[16] Gendreau M, Hertz A, Laporte G. New insertion and postoptimization procedures for the traveling salesman problem. Operations Research 1992;40:1086–94.

[17] Taillard ÉD. Parallel iterative search methods for vehicle routing problems. Networks 1993;23:661–73 The CVRP with distance constraint is also considered.

[18] Volgenant A, Jonker R. The symmetric traveling salesman problem and edge exchange in minimal 1-trees. European Journal of Operational Research 1983;12:394–403.

[19] Rochat Y, Taillard ÉD. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1995;1:147–67.

[20] Toth P, Vigo D. The granular tabu search (and its application to the vehicle routing problem). Technical Report OR/98/9, DEIS, Università di Bologna, Italy, 1998.

[21] França PM, Gendreau M, Laporte G, Müller FM. The *m*-traveling salesman problem with minmax objective. Transportation Science 1995;29(3):267–75.

[22] Golden BL, Laporte G, Taillard ÉD. An adaptive memory heuristic for a class of vehicle routing problems with minmax objective. Computers & Operations Research 1997;24:445–52.

[23] Li LYO. Vehicle routeing for winter-gritting. PhD thesis, Department of Management Science, Lancaster University, 1992.

[24] Eglese RW. Routeing winter gritting vehicles. Discrete Applied Mathematics 1994;48(3):231–44.

[25] Greistorfer P. Computational experiments with heuristics for a capacitated arc routing problem. In: Derigs U, Bachem A, Drexl A, editors. Operations Research Proceedings, 1994. Berlin: Springer; 1994. p. 185–90.
[26] Hertz A, Laporte G, Mittaz M. A tabu search heuristic for the capacitated arc routing problem. Operations Research 2000;48(1):129–35.
[27] Hertz A, Laporte G, Nanchen Hugo P. Improvement procedures for the undirected rural postman problem. INFORMS Journal on Computing 1999;11(1):53–62.
[28] Hertz A, Mittaz M. A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. Transportation Science 2001;35(4):425–34.
[29] Mldadenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.
[30] Lacomme P, Prins C, Ramdane-Chérif W. A genetic algorithm for the capacitated arc routing problem and its extensions. In: Boers EJW editor. Applications of evolutionary computing, Lecture notes in computer science, vol. 2037. Berlin: Springer; 2001. p. 473–83.
[31] Lacomme P, Prins C, Ramdane-Chérif W. Competitive memetic algorithms for arc routing problems. Research Report losi-2001-01, University of Technology of Troyes, France, 2001. Annals of Operations Research, to appear.
[32] Greistorfer P. A tabu scatter search metaheuristic for the arc routing problem. Computers and Industrial Engineering 2003;44(2):249–66.
[33] Beullens P, Muyldermans L, Cattrysse D, Van Oudheusden D. A guided local search heuristic for the capacitated arc routing problem. European Journal of Operational Research 2003;147(3):629–43.
[34] Muyldermans L, Beullens P, Cattrysse D, Van Oudheusden D. The *k*-opt approach for the general routing problem. Technical Report, Center for Industrial Management, Katholieke Universiteit Leuven, 2001. Working Paper 01/18.
[35] Dijkstra E. A note on two problems in connexion with graphs. Numerische Mathematik 1959;1:269–71.
[36] Floyd RW. Algorithm 97: shortest path. Communications of the ACM 1962;5(6):345.
[37] Lenstra JK, Rinnooy-Kan AHG. On the general routing problem. Networks 1976;6:273–80.
[38] Edmonds J, Johnson EL. Matching, Euler tours and the Chinese postman. Mathematical Programming 1973;5:88–124.
[39] Christofides N, Campos V, Corberán A, Mota E. An algorithm for the rural postman problem. Technical Report I.C.O.R. 81.5, Imperial College, 1981.
[40] Corberán A, Sanchis JM. A polyhedral approach to the rural postman problem. European Journal of Operational Research 1994;79:95–114.
[41] Corberán A, Letchford A, Sanchis JM. A cutting plane algorithm for the general routing problem. Mathematical Programming Series A 2001;90(2):291–316.
[42] Benavent E, Campos V, Corberán A, Mota E. The capacitated arc routing problem: lower bounds. Networks 1992;22: 669–90.
[43] Golden BL, DeArmon JS, Baker EK. Computational experiments with algorithms for a class of routing problems. Computers & Operations Research 1983;10(1):47–59.
[44] Li LYO, Eglese RW. An interactive algorithm for vehicle routeing for winter-gritting. Journal of the Operational Research Society 1996;47:217–28.