Computers & Operations Research 40 (2013) 1700–1707

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Heuristics for the traveling repairman problem with profits

T. Dewilde [a,*], D. Cattrysse [a], S. Coene [b], F.C.R. Spieksma [b], P. Vansteenwegen [a,c]

[a] KU Leuven. University of Leuven, Centre for Industrial Management/Traffic & Infrastructure Celestijnenlaan 300 box 2422, 3001 Leuven, Belgium
[b] KU Leuven. University of Leuven, Research Group Operations Research and Business Statistics, Belgium
[c] Ghent University, Department of Industrial Management, Belgium

ARTICLE INFO

Available online 11 January 2013

Keywords:
Traveling repairman
Latency
Tabu search

ABSTRACT

In the traveling repairman problem with profits, a repairman (also known as the server) visits a subset of nodes in order to collect time-dependent profits. The objective consists of maximizing the total collected revenue. We restrict our study to the case of a single server with nodes located in the Euclidean plane. We investigate the properties of this problem, and we derive a mathematical model assuming that the number of nodes to be visited is known in advance. We describe a tabu search algorithm with multiple neighborhoods, we test its performance by running it on instances from the literature and compare the outcomes with an upper bound. We conclude that the tabu search algorithm finds good-quality solutions fast, even for large instances.

1. Introduction

Imagine a single server, traveling at a constant speed. There are n locations given, each with a profit p_i, $1 \le i \le n$. At $t=0$, the server starts traveling and collects a revenue $p_i - t_i$ at each visited location, where t_i denotes the server's arrival time at location i. Not all locations need to be visited. The problem is to find a travel plan for the server that maximizes the total revenue. Notice that when the profits are very large, i.e., $p_i > t_i$, $\forall i$, the problem comes down to finding a tour minimizing the sum of the waiting times; this problem is known as the minimum latency or traveling repairman problem. The problem studied in this paper is the traveling repairman problem with profits (TRPPs). In particular, we perform a computational study of the TRPP in the Euclidean plane.

1.1. Motivation

The TRPP occurs as a routing problem in relief efforts. For example, consider the following situation. In the aftermath of a disaster like an earthquake, there are a number of villages that experience an urgent need for medicine. The sooner the medicine gets to a village, the more people can be rescued. Since the cost of transport is negligible compared to the value of a human life, rescue teams are only concerned with the total number of people that can be saved. Assume that at location i, p_i people are in need of the medicine, and that every instance of time, there is one of them dying. Suppose also that we have one truck available. With t_i denoting the arrival time of the truck at location i, the number of people that will survive equals $p_i - t_i$. Thus, the goal of the rescue team is to maximize $\sum_i (p_i - t_i)$, where the sum runs over all the visited locations. Note that from the moment that t_i becomes larger than p_i for an unvisited location i, that location will not be visited anymore. This situation is described in [6] and is equivalent to the TRPP.

Another, more theoretical, motivation concerns the k-traveling repairman problem (k-TRP). The k-TRP is a traveling repairman problem with multiple servers. All clients need to be visited such that the latency, i.e., the sum of the waiting times, is minimized. Observe that no profits are considered in this problem. As explained in Coene and Spieksma [7], one potential way of solving such a problem is a set-partitioning approach where an integer programming model is built, using a binary variable x_{rk} for each set of clients; this variable is equal to 1 if route $r \in \{1,\ldots,R\}$ is served by server $k \in \{1,\ldots,K\}$, with R and K the number of feasible routes and the number of servers, respectively. Let c_{rk} be the latency of a feasible route r served by server k. With this notation, the set-partitioning approach looks as follows:

$$\text{Minimize} \quad \sum_{r=1}^{R} \sum_{k=1}^{K} c_{rk} x_{rk}$$

$$\text{subject to} \quad \sum_{r:i \in r} \sum_{k=1}^{K} x_{rk} = 1 \quad \text{for } i = 1,\ldots,n,$$

$$\sum_{r=1}^{R} x_{rk} \le 1 \quad \text{for } k = 1,\ldots,K,$$

$$x_{rk} \in \{0,1\} \quad \text{for } r = 1\ldots,R \quad \text{and} \quad k = 1,\ldots,K.$$

* Corresponding author. Tel.: +32 16 32 25 66.
E-mail address: Thijs.Dewilde@cib.kuleuven.be (T. Dewilde).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.01.003

When applying a branch-and-price approach to the resulting integer program, it can be observed that the so-called pricing problem in this approach is exactly the TRPP, where the dual variables play the role of profits.

Other applications of routing problems with time-dependent revenues are described in Coene and Spieksma [7], Erkut and Zhang [9], Lucena [13], and Melvin et al. [14], which deals with a problem occurring in "multi-robot routing".

1.2. Literature

Several problems are closely related to the traveling repairman problem with profits (TRPP). The TRPP has similarities with the traveling salesman problem (TSP) [3]. However, contrary to the TSP, in the TRPP not all the nodes need to be visited. Further, an optimal TRPP-solution is a path whose course is influenced by the depot location and may contain intersections. Notice that the latter is always sub-optimal for the Euclidean TSP.

Routing problems with profits are, e.g., the TSP with profits (TSPP) [10] and the orienteering problem (OP) [21]. In the latter, a subset of nodes should be selected in order to maximize the profit under a time-constraint. As in the TRPP, an optimal solution may leave some nodes unvisited. Both the total profit and the distance traveled are inserted in the objective function of the TSPP. The TRPP, however, differs in the sense that revenues are time-dependent.

Problems with time-dependency are relevant in many cases. See [13] for the time-dependent traveling salesman problem (TDTSP). In the TDTSP, the travel time between two vertices depends on the arrival time of the server. This is different from the TRPP where the travel time between vertices is constant, but the profit depends on the arrival time. Tang et al. [20] introduced the multiple tour maximum collection problem with time-dependent rewards; a problem in which jobs are to be scheduled over multiple days and in which the reward corresponding to a job is time-dependent. For this problem, the goal is to find multiple tours, and rewards depend on which tour (i.e., which day) the job is assigned to; this is different from the TRPP, where rewards depend on the position of the job in the tour and not all jobs need to be scheduled.

The TRPP is a generalization of the traveling repairman problem (TRP) [5], i.e., an instance of the TRPP with extremely high profits is also an instance for the TRP with the same optimal solution. In the TRP (also known as the minimum latency problem or the delivery man problem), a single server needs to visit all nodes such that the total latency is minimized. In a classical paper by Afrati et al. [2], it is shown that the TRP on the line can be solved in polynomial time by dynamic programming. This result was generalized to the TRPP on the line by Coene and Spieksma [7]. Since the TRP is NP-hard for more general metric spaces, see the argumentation given in [5], and since the TRPP is a generalization of the TRP, we conclude that the TRPP for these general metric spaces, among which the Euclidean plane, is NP-hard.

As far as we know, no computational studies have been performed for the TRPP. In [7], the TRPP on the line is being solved in polynomial time by a dynamic programming algorithm. No other results are known.

Exact algorithms and approximation algorithms for the TRP have been described in Ausiello et al. [4], Goemans and Kleinberg [12], and Wu et al. [22]; a metaheuristic for the TRP is described in the contribution of Salehipour et al. [18]. An extension of the TRP, the cumulative VRP, is dealt with by Ngueveu et al. [16]. Their memetic algorithm can also be used to solve TRP-instances; this is done for comparison in [18]. Note that the algorithm presented in [16] uses a tour-splitting procedure which has no effect in the single-vehicle case of the TRP. For a review of the metaheuristics for other related problems we refer to Feillet et al. [10], Vansteenwegen et al. [21], and the references contained therein. A general description of some metaheuristics, including the ones that are used in this paper, is given in [11] and [19].

This paper is structured as follows. In the next section, the TRPP is described in detail and a mathematical model is given. A tabu search algorithm is presented in Section 3. The data sets are introduced in Section 4 and the computational results are discussed in Section 5. In Section 6, we compare our results for the TRP with the results from the literature. The conclusions of this paper are summarized in Section 7.

2. Mathematical model

Consider a complete undirected graph $G=(V,E)$, where $V=\{0,1,\ldots,n\}$ is the node set, and E is the set of edges. Each node $i\neq 0$ has an associated profit p_i. There is a single server located at node 0, the depot. The time it takes the server to travel from node i to node j is defined by $d_{i,j}$, with, $d_{i,j}$ satisfying the triangle inequality for all i,j. Further, we assume that the time to serve a node is negligible. If the server arrives at node i at time t_i, a revenue of p_i-t_i is collected. As a consequence, an optimal tour will not contain a node i with $p_i\leq t_i$. The goal of the TRPP is to select an ordered subset of nodes such that visiting them one by one maximizes the sum of all the revenues. Each node can only be visited once, and the server does not need to return to the depot.

We now derive a mathematical model for this problem when the number of nodes to be visited, k, is given. Thus, k is the number of nodes whose revenue is collected. Notice that k is just an integer that contains no information about which nodes are visited. The optimal solution for the original problem, with k a variable, can then be found by solving the model for each value of k and selecting the best one; we will come back to this issue at the end of this section. Let $K=\{1,2,\ldots,k\}$.

For each $i\in V, j\in V_0=V\backslash\{0\}$, and $\ell\in K$, we define the variable $y_{i,j,l}$ as follows:

$$y_{i,j,\ell}=\begin{cases}1 & \text{if edge } (i,j) \text{ is used as the } \ell\text{th edge,}\\ 0 & \text{else.}\end{cases}$$

This definition says that if $y_{i,j,\ell}=1$ then (i,j) is the ℓth edge of the path. Hence i is the $(\ell-1)$th and j is the ℓth node that is visited. The depot is node 0. Observe that if $y_{i,j,\ell}=1$, $d_{i,j}$ is counted $k+1-\ell$ times in the total latency. Hence

$$\sum_{i:i \text{ is visited}} t_i=\sum_{\{(i,j,\ell)|y_{i,j,\ell}=1\}}(k+1-\ell)d_{i,j}. \tag{1}$$

Now the mathematical model can be constructed.

With k, the number of nodes visited, given, the mathematical model is the following:

$$\text{Maximize}\quad \sum_{i\in V}\sum_{j\in V_0}\sum_{\ell\in K}(p_j-(k+1-\ell)d_{i,j})y_{i,j,\ell} \tag{2}$$

$$\text{subject to}\quad \sum_{i\in V}\sum_{\ell\in K}y_{i,j,\ell}\leq 1\quad \forall j\in V_0, \tag{3}$$

$$\sum_{i\in V}\sum_{j\in V_0}y_{i,j,\ell}=1\quad \forall \ell\in K, \tag{4}$$

$$\sum_{i\in V}y_{i,j,\ell}-\sum_{i\in V_0}y_{j,i,\ell+1}=0\quad \forall j\in V_0,\ \forall \ell\in K\backslash\{k\}, \tag{5}$$

$$\sum_{j \in V_0} y_{0,j,1} = 1, \tag{6}$$

$$y_{i,j,\ell} \in \{0,1\} \quad \forall i \in V,\ \forall j \in V_0,\ \forall \ell \in K. \tag{7}$$

The objective function (2) sums the difference between the profit of a node and the number of times the edge preceding that node is counted in the total latency, see (1). The first set of restrictions makes sure that each node can be visited at most once (3). The second set dictates that k nodes different from the depot must be visited; for each $\ell = 1,2,\ldots,k$ the server has to travel from a node $i \in V$ to a node $j \in V_0$ (4). Constraints (5) ensure the connectivity; the departure from the depot is arranged by (6). Finally, all $y_{i,j,\ell}$ must be binary (7). This model is used in Section 5 for obtaining an optimal solution (or the LP-relaxation) for some of the considered instances using IBM ILOG Cplex 10.1. As the TRPP is shown to be NP-hard, it is expected that the solver will be shortcoming for solving large instances using our mathematical model. That is why a metaheuristic and an upper bound, different from the LP-relaxation, are introduced in Section 3.

Recall that in this model the value of k is given. However, in the TRPP, k is a decision variable and should be determined by the model itself. It is not difficult to introduce k as a variable in the model, see Dewilde [8]. However, preliminary results [8] show that this leads to a weaker LP-relaxation. Indeed, solving the LP-relaxation of (2)–(7) for all values of k, yields an optimal relaxed solution for the TRPP with an integer value of k. The LP-relaxation of a model including k as an integer, may yield a relaxed solution with a non-integer value for k. On the other hand, the TRPP is a hard problem, even for a given value of k, and it will be shown next that determining the best value for k is not straightforward, apart from solving the above model for each value of $k \leq n$.

Before doing so, let us first introduce some notation. Define k^* as the optimal number of visited nodes and f^* as the global optimal objective value. Define $f(k)$ as the optimal objective value for which the solution visits exactly k nodes, hence $f^* = f(k^*)$. As mentioned above, we will now show that the mathematical model needs to be solved for each value of $k \leq n$ in order to find k^* and hence the global optimum. Therefore we will demonstrate that (i) $f(k)$ in function of k is not unimodal and (ii) adding nodes to an instance may result in a decreasing value for k^*. Let us first go into (i). It holds that when the server is forced to visit one node more than the k^* nodes which lead to f^*, this results in an inferior solution. Intuitively one may think that the further k lies from the optimal number of visited nodes, k^*, the worse the objective value will be. In other words, our intuition may tell us that for any $k \geq k^*$ we have that $f(k^*) \geq f(k) \geq f(k+1)$, and analogously for any $k \leq k^*$. However, this is not always true. To show this, consider the network with six collinear points given in Fig. 1. The numbers between brackets are respectively the location along the axis and the profit. So the leftmost node, node 5, has as coordinate -50 and its profit equals 54.

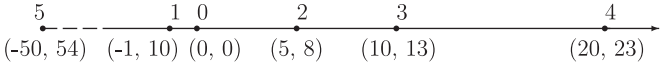

Fig. 1. Network with six collinear points.

When we solve this instance to optimality for $k = 1,2,\ldots,5$, i.e., when we force the solution to visit exactly k nodes, we find the results depicted in Fig. 2(a). It can be seen that when solving the mathematical model for $k=2$, the resulting path is $\langle 0,1,5 \rangle$ with total revenue 13, for $k=3$ the optimal path becomes $\langle 0,1,2,3 \rangle$ and has revenue $f(k) = 11$, whereas forcing k to be 4, the solution is $\langle 0,1,2,3,4 \rangle$ with objective value 12. If we force the solution to visit all five nodes, the solution becomes $\langle 0,1,2,3,4,5 \rangle$ and a negative profit is collected at the last node from the path causing the total revenue to become negative. From the figure it is clear that $k^* = 2$. More importantly, the non-unimodality of this graph shows that $f(k)$ can have multiple local optima, which suggests that, in order to find k^* for a particular instance, model (2)–(7) has to be solved for each $k = 1,\ldots,n$.

The second property (ii) that can be deduced from this example deals with adding a node to an instance. If an extra node is added to a data set, our intuition may tell us that the optimal number of visited nodes will be the same or larger than before adding that node. However, this is not always true. Clearly, by adding a new node to an instance, the optimal value cannot decrease. But nothing can be said about the optimal number of visited nodes of this new instance as witnessed by the given example. Define $I_m : m \leq n$, as the instance consisting of the first m nodes of the network. The number of nodes in the optimal solution for instance I_m is $k^*(I_m)$. The results for the value of $k^*(I_m)$ for the network of Fig. 1 are given in Fig. 2(b). This example indicates that knowing $k^*(I_m)$ for a certain value of m does not give any information about $k^*(I_{m'})$ with $m' > m$. Again, we can only conclude that, to find k^*, the model (2)–(7) has to be solved for each $k = 1,\ldots,n$. Notice that the observations above already hold in the case of a line metric; the TRPP on the line is solvable in polynomial time [7].

3. Metaheuristic method

In this section, a metaheuristic for the TRPP is presented. First, we discuss an algorithm to build a non-trivial solution which will then be systematically improved by a tabu search algorithm. We define the *trivial solution* as the path $\langle 0 \rangle$, i.e., the situation in which the server does not leave the depot. In the construction phase, we start from the trivial solution and

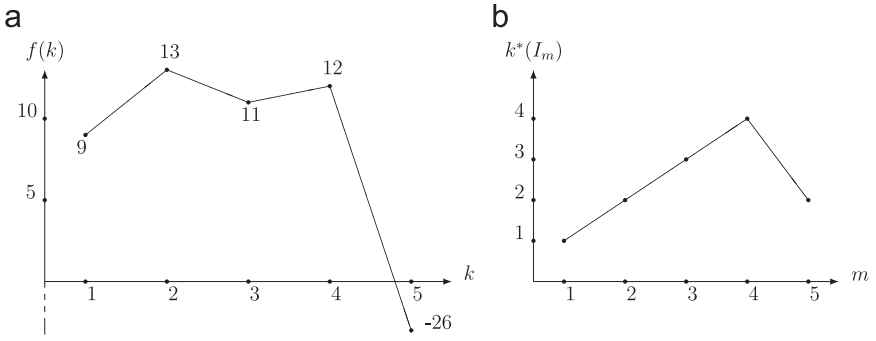

Fig. 2. Solution results for the network with six collinear points.

add a node in each step, obtaining a new solution. This process is discussed in the next section. The improvement phase is dealt with in Section 3.2, where some neighborhoods for local search are presented. These neighborhoods are integrated in a tabu search metaheuristic.

3.1. Construction phase

Consider a partial path P, and define the set $\overline{V}$ as the set of all non-visited nodes, $\overline{V} \subseteq V_0$. In order to improve the partial path P, a node from $\overline{V}$ can be added. This process requires two decisions: which node to insert and where to place it in the path. Naturally, two factors influence these choices: the profit of the nodes and the additional latency incurred by inserting that node.

Let $d_{i,j}$ and p_j be as before. Then, for each $i \in V \backslash \overline{V}$ and $j \in \overline{V}$ we define $ratio_{i,j}^m$ as follows:

$$ratio_{i,j}^m = \begin{cases} \dfrac{1}{d_{i,j}} & \text{if } m = 0, \\ p_j \cdot \left(\dfrac{1}{d_{i,j}}\right)^m & \text{if } m = 1, \ldots, 10. \end{cases} \tag{8}$$

In this way, the parameter m determines the impact of p_i and $d_{i,j}$ on the ratio.

The construction method that is used in this paper is based on the insertion. First, it is decided which node to insert. For each value of $0 \le m \le 10$, two sets of candidates are considered for insertion. These candidates are selected in the following way: for each value of m, the node $j^* = \arg\max_{j \in \overline{V}} ratio_{i,j}^m$, with i the last added node (first set of candidates) or the last node in the path (second set of candidates), is selected, yielding $2(m+1)$ candidates for insertion. The place of insertion is determined based on the improvement in score (i.e., total collected revenue) by adding this node. For each candidate, we consider all possible insertions and select the best one. If the best solution over all candidates yields an improvement, the node is added. When no improvement is obtained, the algorithm stops and the best found solution is returned.

It has been pointed out by many authors that an insertion based construction method performs better than the nearest-neighbor heuristic for the Euclidean TSP [17], of course this gives no guarantee for the performance here. In Table 1, we compare the objective function values of our insertion based construction method with nearest-neighbor solutions. To do so, we define the improvement in percent as

$$\text{improv}(\%) = 100 \cdot \frac{\text{insertion} - \text{nearest-neighbor}}{\text{nearest-neighbor}}. \tag{9}$$

The instances described in Section 4 are used for this comparison. In both cases, the computation times are similar and very small. From the results in Table 1, we can confirm the results from the literature for the TRPP.

3.2. Improvement phase

This section describes a tabu search metaheuristic for the TRPP. The insertion based algorithm from the previous section is used as input. A tabu search metaheuristic starts from a given solution. By searching neighboring solutions, it tries to improve the current solution. Our algorithm works with multiple neighborhoods. Next, we define the moves and the corresponding neighborhoods. Then, the tabu search procedure is explained in Section 3.2.2.

Table 1
The average improvement of the insertion based construction method over the nearest-neighbor heuristic.

n	10	20	50	100	200	500
improv(%)	4.68	2.16	1.32	0.98	0.32	1.11

3.2.1. Neighborhoods

As mentioned before, multiple neighborhoods are explored to improve the solution from the construction phase. The objective of the TRPP is to maximize total collected revenue, which is based on the profits that decrease over time. Hence, improving a solution can be done by altering the collection of visited nodes, or by changing the visiting sequence. A first set of three different neighborhoods is obtained by moves that alter the subset of selected nodes: deletion, insertion, and replacement [10]. A second set of seven different neighborhoods is obtained by moves that alter the visiting sequence. These are the well-known swap and swap-adjacent, 2-opt, and a generalized Or-opt [1]. The choice for Or-opt is justified by the fact that the visiting order of the moved subpath is not reversed, while the new solution differs enough from the previous to circumvent local optima where other moves might end up. We use the term "generalized Or-opt" because we allow to move subpaths with more than three nodes. Further, there is move-up (down), which consists of shifting a node up (down) the path, and a special type of move-up, namely remove–insert [18]. In this move the node with the largest between-distance of a given node is removed and back inserted at the end of the path.

Although swap-adjacent and remove–insert are special cases of swap and move-up, respectively, they are used separately. This is because they have linear complexity, while move-up (down) and swap have a neighborhood of size $\mathcal{O}(n^2)$. Hence, separating these moves can speed up the algorithm. Note that the same can be said about move-up (down) and Or-opt which has a neighborhood of size $\mathcal{O}(n^3)$.

In each iteration of the tabu search algorithm (see below), a move will be selected according to the principles of a variable neighborhood descend heuristic (VND) [11,19]. This means that the 10 neighborhoods will be searched through one by one, in the sequence of Fig. 3. Whenever an improving move is detected, the best solution from the corresponding neighborhood is chosen as next solution. Only in the case that there is no better solution in a neighborhood, the next neighborhood will be explored. First, the neighborhoods that alter the sequence of the path are explored, then those that alter the set of nodes. The last neighborhood explored is the one obtained with Or-opt, because a larger sized neighborhood may help escaping from a local optimum. The choice for this sequence is justified by the fact that re-optimizing the visiting sequence after having altered the set of selected nodes will yield better solutions. Notice that, to save computation time, not all neighborhoods will be explored exhaustively, we will come back to this in the description of the tabu search algorithm.

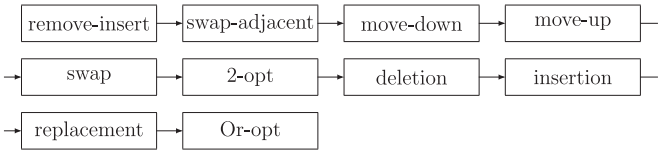

Fig. 3. Sequence of the moves.

3.2.2. Tabu search

The neighborhoods discussed above are inserted in a tabu search metaheuristic (TS) [11,19], which will guide the search for improving solutions. The basic idea of tabu search is to avoid repetition of solutions and to use steepest ascend combined with mildest descend to escape from local optima.

The algorithm starts with the solution obtained from the insertion based construction method and consists of four main steps which are repeated until a stop criterium is met. The first step, which is the main body of the algorithm, consists of a variable neighborhood descend heuristic (VND). In the second step, a list of promising solutions is built up and serves as input for the intensification phase (step 3). In order to explore the entire solution space, a diversification phase is added as fourth step. A pseudocode of the complete algorithm is given in Algorithm 1. The actual values of the parameters are determined based on preliminary experiments [8].

In the remaining of this section, some more details about the four steps of our algorithm are given.

Step 1: In this step, the 10 neighborhoods of Fig. 3 are explored following the principles of variable neighborhood descend (VND). The observation that the selected moves involve only nodes that are close together in the solution path led to a restriction on the larger neighborhoods. More specifically, for move-up (down), swap, 2-opt, and Or-opt the maximum number of visited nodes in the path between the move-determining attributes is set to 200. Using this value, no promising move is missed and a gain in computation time is realized. In each iteration, the best neighboring solution is accepted if it is non-tabu and improving, or tabu but globally improving (aspiration).

Due to the use of different neighborhood structures and the frequency with which they deliver a new solution, three tabu lists are used. The first list corresponds to deletion, insertion, and replacement moves. A move of the type remove–insert, swap (adjacent), or move-up (down) is captured in the second tabu list, and the third tabu list is for 2-opt and Or-opt moves. After each move only the tabu list of the corresponding neighborhood is updated. The reverse move becomes tabu for a number of iterations that is determined dynamically. For tabu lists one and two, the tabu tenure is a random number between 10 and 20, and for the tabu list of 2-opt and Or-opt the tabu tenure varies between 3 and 7, again at random. The lower bound is determined based on the regularity of the tabu list updates and the upper bound is chosen such that some diversification in tabu tenure is available. After the reverse move is added, the remaining tabu time of the other moves in that tabu list is reduced by one.

Step 2: If a local optimum is reached in step 1, the second step starts in which a list L of promising solutions is built up. Let k be the number of nodes visited in the current solution, then the local optimum is added to the list of promising solutions if it is good and diverse enough, i.e., if its objective value lies within 5% of the best found solution and at least $\min(100,k)$ moves are made since the previous addition to L. If the current solution becomes the fourth element of L, the algorithm proceeds to step 3. Otherwise, to escape from a local optimum, a ''shaking'' similar to what is done by Mladenovic and Hansen [15] in their skewed VNS algorithm, is performed before the algorithm returns to step 1. The ''shaking'' here corresponds to the best non-tabu move that either changes the subset of selected nodes, or incurs a decrease in objective value smaller than $10 \cdot lat/k$, with lat the total latency of the previous solution and k the number of visited nodes. The values of the parameters used in this step are chosen based on their impact on the whole algorithm during preliminary computations.

Step 3: If the promising solutions list L is full, the intensification phase starts. First, the three tabu lists are cleared. Then, one by one, the solutions of L are used as start solution for a full neighborhood VND without tabu moves. This corresponds to step 1 without the restriction on the size of the neighborhoods and without any tabu list. When a (mostly new) local optimum is found for all elements of the list of promising solutions, the algorithm proceeds to step 4 in which a diverse solution to re-initialize the search is created.

A small value for the size of L enhances more calls to the intensification and diversification steps (steps 3 and 4). But, due to the more frequent diversification, the search can be directed to another area of the solution space without the previous area being completely explored. On the other hand, when the value of $|L|$ is large, it takes longer to build L, such that less intensification and diversification runs are performed. They are, however, computationally more intensive due to the size of L. As a compromise, the size of four is selected.

Step 4: The fourth step begins with updating the attribute matrix M, whose entries $[M]_{i,j}$ represent the number of times edge (i,j) was present in an element of the promising solutions list. This matrix allows to guide the search towards an unexplored part of the solution space by penalizing frequently used edges and nodes using the function $\tilde{f}$ instead of f as objective function:

$$\tilde{f}(x) = f(x) - 0.002 \cdot f(x^*) \cdot \sum_{(i,j) \in E(x)} [M]_{i,j} - 0.01 \cdot \sum_{i \in V(x)} \left(p_i \cdot \sum_{j \in V} [M]_{i,j} \right), \tag{10}$$

where $E(x)$ and $V(x)$ denote the edge and node set of solution x, respectively, and p_i is the profit of node i. Using edge (i,j) incurs a penalty of 0.2% of the best found solution value multiplied by $[M]_{i,j}$, and the profit of each visited node, p_i, is decreased by 1% for each time it was part of a promising solutions. These percentages are selected because they gave the best results during preliminary computations in which the edge penalty was allowed to deviate in steps of 0.001 and the node penalty in steps of 0.002.

To find a new and distinct solution, the algorithm returns to step 1 but with the restriction that for the first $\max(100,2k)$ iterations the objective function f is replaced by $\tilde{f}$. During this process, the tabu lists are built up from scratch to prevent a quick return towards the previous solutions.

During a preliminary study [8], it was noticed that, for some smaller instances, the algorithm found many near-optimal solutions which all contained parts of the optimal solution without finding the optimal solution itself. To push the search to combine parts of high quality solutions by altering the sign of the penalties in each fourth diversification phase, this was avoided and better results were achieved.

The last aspect to discuss is the stop criterium of the tabu search algorithm for the TRPP. A balance must be made between computation time and efficiency. The number of consecutive non-improving steps (n-impro) and a maximum computation time (compT) determine the stop criterium, see Table 2.

Table 2
The stop criterium of the tabu search algorithm.

n	n-impro	CompT (s)
≤ 100	20,000	
≤ 200	50,000	1000
≤ 500	100,000	2000

Algorithm 1. Tabu search algorithm for the TRPP.

input: objective functions f and $\tilde{f}$, neighborhoods $N_i(x), i=1,\ldots,10$, initial solution x, $x^* \leftarrow x$, empty tabu lists, $L \leftarrow \emptyset$, $M \leftarrow 0$
while stop criteria not met **do**
step 1: (main body, VND)
  for $i=1 \rightarrow 10$ **do**
    $x' \leftarrow \arg\max_{x'' \in N_i(x)} f(x'')$
    if ($f(x') > f(x)$ and x' is not tabu) or $f(x') > f(x^*)$ **then**
      $x \leftarrow x'$
      $i \leftarrow 1$
      Update tabu list
      if $f(x') > f(x^*)$ **then**
        $x^* \leftarrow x'$
    else
      $i \leftarrow i+1$
step 2: (Built up promising solutions list L)
  if $f(x) > 0.95 \cdot f(x^*)$ and at least min(100,k) moves, k being the number of visited nodes in x, are made since the previous solution is added to L **then**
    $L \leftarrow L \cup \{x\}$
    if $|L| == 4$ **then**
      go to step 3
    else

$$x \leftarrow \arg\max \left\{ f(x'') \;\middle|\; \begin{array}{l} x'' \in \bigcup_{i=1}^{10} N_i(x),\ x'' \text{ is not tabu, and} \\ 7 \le i \le 9 \text{ or } f(x'') > f(x) - 10 \cdot lat/k. \end{array} \right\}$$

      Update tabu list
      go to step 1
step 3: (Intensification)
  for $j=1 \rightarrow 4$ **do**
    Perform step 1 without the tabu search update and without neighborhood restrictions with the jth element of L as start solution
go to step 4
step 4: (Diversification)
  Update attribute matrix M
  Clear all tabu lists
  go to step 1 using $\tilde{f}$ instead of f for the first max(100,2k) iterations

3.3. Upper bound

Since the TRPP in the Euclidean plane is NP-hard (see Section 1), the mathematical model can only be used to solve small instances. Thus, to assess the quality of the solutions found by tabu search, an upper bound, different from the LP-relaxation, is required. We describe here a simple bound. Assume that the number of visited nodes, k, is known. In order to get a lower bound for the latency in case k nodes are visited, we use the k-minimal spanning tree (k-MST). Since solving a k-MST is again an NP-hard problem, we use the minimal k-forest to approximate this. The minimal k-forest of a graph is the subgraph containing the $k-1$ shortest edges that do not form a circuit. Next, each edge of the k-forest is assigned a multiplicity. The longest edge gets 1, the second longest 2, etc., until the shortest edge gets k. The sum of the distances weighted with the corresponding multiplicities is then a lower bound for an optimal solution to the k-MST [18].

By summing the k largest profits, we get an upper bound for the collected profits. The difference of this upper bound and the lower bound for the k-MST gives an upper bound for the TRPP, under the assumption that k is known. Taking the maximum over $k=1,\ldots,n$, leads us to an upper bound for the TRPP.

Table 3
Comparison of the results of the mathematical model (2)–(7).

n	Cplex	LP-relaxation		Construction	Tabu search		Upper bound
	Time (s)	Gap (%)	Time (s)	Gap (%)	Gap (%)	Time (s)	Gap (%)
10	1	(−)3.15	0	0.21	0	0	(−)19.51
20	84	(−)5.36	1	1.87	0	0	(−)11.87
50			145	6.59[a]	5.22[a]	10	(−)2.32[a]

[a] Gap with the LP-relaxation.

4. Instances

The TRPP has not been studied computationally before, and there are no specific data sets available. However, as mentioned before, the TRPP is a generalization of the TRP, for which there are data sets available. That is why we used the data sets introduced in [18] and are also used in [16]. These data sets are constructed at random and the problem size ranges between 10 and 500 nodes, apart from the depot node. For each problem size, 20 instances are created. To adapt these instances to the TRPP, we allocated a profit to each node different from the depot. This is done using the uniform distribution in the interval (L,U] with L the minimum distance between the depot and a node, i.e., $L=\min_i d(0,i)$, and $U=n/2 \cdot \max_i d(0,i)$, where n equals the number of nodes. In this way, a good quality solution contains about 80% of the nodes.

5. Results

In this section we will discuss the computational results. First, a comparison between the exact results, the tabu search results, and the upper bound from Section 3.3 is given for small data sets. Second, the upper bound is used to measure the performance of tabu search on larger instances.

The first results are presented in Table 3. The first column shows the average computation time for the exact solutions, found by solving the mathematical model from Section 2 using IBM ILOG Cplex 10.1 run on a DELL Optiplex 760, Intel(R) Core(TM) 2 Duo 3.00 GHz, 4.00 GB RAM, 64-bit Operating System. This is the total time required to solve the model for all $k=0,\ldots,n$, with k the number of visited nodes. The other columns present (i) the average gap between the LP-relaxation's value, computed as the maximum of the relaxation of (2)–(7) for all $k=0,\ldots,n$, and the optimal solution's value, and the time needed to compute the LP-relaxation, (ii) the average gap between the value of the solution found in the construction phase and the optimal solution's value, (iii) the average gap between the value of the solution found by the tabu search algorithm and the optimal solution's value, and (iv) the average gap between the value of the upper bound and the optimal solution's value, respectively. The latter three are implemented in C++ and run on the same processor as specified above. The gap is computed as follows:

$$\text{gap}_X(\%) = 100 \cdot \frac{(\text{exact solution}) - X}{\text{exact solution}}. \tag{11}$$

In the case of $n=50$, we are not able to compute exact results since solving the mathematical model for some $k \le n$ requires too much computation time. In this case the gap is computed with respect to the LP-relaxation. When n gets larger, Cplex is not able to find a solution anymore due to memory restrictions. Computation times for the construction phase and upper bound are neglectable, and therefore not included in the tables.

Table 4
Comparison of the results of the metaheuristic.

n	Construction	Tabu search	
	Gap (%)	Gap (%)	Time (s)
10	16.27	16.10	0
20	12.19	10.54	0
50	8.71	7.37	10
100	6.09	4.83	105
200	4.32	3.15	1005
500	10.87	8.96	1999

Table 5
Results for TRP instances solved with the mathematical model (3)–(7) and (12).

n	Cplex	LP-relaxation		Tabu search	
	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
10	0	9.18	0	0	0
20	20	17.22	0	0	0
50			10	−35.66[a]	61

[a] Gap with the LP-relaxation.

From Table 3, it is clear that tabu search is able to find an optimal solution for all instances when $n=10$ or 20. When $n=50$, we see that on average, the gap with the LP-relaxation is 5.22%. Next, we can see that TS needs much less time than Cplex. Finally, the upper bound gives worse results than the LP-relaxation, but it needs much less time.

In Table 4, the average gap between the value of the upper bound and the value of the construction phase solution (first column) or the tabu search solution (second column) is given. The gap is defined analogue as in (11). For TS, the computation times are also given.

A first thing that can be noted from the results in Table 3 is the relatively high value of the gap between the exact solution and the upper bound. This shows the relative poor performance of our, indeed simple, upper bound. However, for the larger instances in Table 4, the gap with the tabu search solutions varies between 3% and 9%, which is rather good for such a crude upper bound. From this it can be concluded that, for the instances solved, tabu search finds near-optimal solutions, even for the large instances. In Table 4, it can also be seen that the difference in gap between the construction phase solution and the tabu search solution is rather small. This indicates that the insertion based construction phase returns good quality solutions fast. Although the improvement of tabu search upon these construction phase solutions is not too large, it is significant, since the solutions are within 9% of the optimum. Notice that for the instances with $n=10$ and 20, tabu search finds the optimal solution and hence cannot obtain a larger improvement.

6. TRP

As we mentioned in the first section, the TRPP is a generalization of the TRP. It follows that any solution method for the TRPP can be used to solve instances of the TRP. In this section we use our mathematical model and tabu search heuristic to solve instances of the TRP and compare this with heuristics described in the literature for the TRP [18] or for an extension of the TRP, as there is [16]. We first solve our instances with up to 50 customers using the mathematical model (3)–(7). Larger instances cannot be solved to optimality in reasonable time. In the TRP all customers are visited, thus $k=n$, and all profits are equal to 0, such that the objective function becomes the following:

$$\min \sum_{i \in V} \sum_{j \in V_0} \sum_{\ell \in K} ((k+1-\ell)d_{i,j}) y_{i,j,\ell}. \quad (12)$$

Table 6
Comparison of our tabu search algorithm applied on the TRP with other TRP-metaheuristics from the literature.

n	Ngueveu et al. [16]		Salehipour et al. [18]		Tabu search	
	Improv (%)	Time (s)	Improv (%)	Time (s)	Improv (%)	Time (s)
10	2.43	0	2.44	0	2.43	0
20	10.11	0	9.86	0	10.11	0
50	9.36	1	9.67	3	9.29	61
100	11.95	62	11.56	101	11.61	503
200	12.81	619	11.33	3741	11.69	1014
500	13.85	10,698	8.11	91,471	8.07	2011
Ave	10.09	1897	8.83	15,886	8.87	598

The results are shown in Table 5. The gap is computed using (11). The solution times are smaller than in Table 3 which is normal as the instances only need to be solved for a single value of k. The linear relaxation gap and its computation time are, on average, larger for instances without profits. For $n=10$ or 20, tabu search finds the optimal solution for all instances.

Salehipour et al. [18] compare their metaheuristic for the TRP with the memetic algorithm of Ngueveu et al. [16] which is initially developed for the cumulative VRP. The comparison is based on the percentage improvement of the metaheuristics with respect to the nearest-neighbor heuristic. Using our tabu search heuristic designed for the TRPP and initiated with the insertion based construction phase solution, we can apply the same. The obtained results are summarized in Table 6. In this table, we compare the average improvement of each metaheuristic with respect to the nearest-neighbor solution. The values in the columns improv are computed analogue to (9). The computation times of Ngueveu et al. are obtained using a Pentium D processor at 3.4 GHz with 1 GB RAM while Salehipour et al. run their algorithms on a 2.4 GHZ Pentium 4 computer with 512 MB RAM.

By comparing the numbers in the improv-columns, it becomes clear that, although not designed for it, our tabu search algorithm performs nearly as good as already existing TRP-metaheuristics. In particular, computation times of our tabu search grow quite moderate compared to the other approaches while the solution quality is comparable.

7. Conclusion

We have studied the traveling repairman problem with profits (TRPP). In this problem a server has to visit a subset of nodes in order to maximize the total collected revenues which are declining over time. After motivating this problem and reviewing the related literature, we develop a mathematical model in which we make the assumption that the number of visited nodes is known in advance. Using an example, we find that it is not straightforward to determine this number optimally. As our main contribution, we propose a tabu search algorithm with multiple neighborhoods. We have implemented this method, and we tested the performance of this metaheuristic, comparing it to a quite crude upper bound. The computational results show that the tabu search algorithm is able to find optimal solutions for small instances in a reasonable amount of time. For larger instances the optimal solution is not known, but the metaheuristic is able to reduce the gap with the upper bound to 3–9%. Even for TRP instances, our algorithm significantly improves the initial

solution and performs nearly as good as state-of-the-art algorithms.

Further work: In this paper we studied the TRPP with a single server. However, the algorithm can be extended for the multi-server case. This is an interesting topic for further research. In the Introduction we mentioned how the profit-based latency setting can be relevant in relief efforts, it would be worthwhile to study this link in more detail.

Acknowledgment

Research funded by a Ph.D. grant of the Agency for Innovation by Science and Technology (IWT). Dr. S. Coene is a post-doctoral research fellow of the "Fonds Wetenschappelijk Onderzoek - Vlaanderen (FWO)".

References

[1] Aarts E, Lenstra JK, editors. Local search in combinatorial optimization. New Jersey: Princeton University Press; 1997.
[2] Afrati F, Cosmadakis S, Papadimitriou CH, Papageorgiou G, Papakostantinou N. The complexity of the travelling repairman problem. Informatique Théorique et Applications 1986;20(1):79–87.
[3] Applegate DL, Bixby RE, Chvátal V, Cook WJ. The traveling salesman problem: a computational study. New Jersey: Princeton University Press; 2006.
[4] Ausiello G, Bonifaci V, Leonardi S, Marchetti-Spaccamela A. Prize-collecting traveling salesman and related problems. In: Gonzalez TF, editor. Handbook of approximation algorithms and metaheuristics. Boca Raton: Chapman & Hall, CRC; 2007. p. 40.1–13.
[5] Blum A, Chalasani P, Coppersmith D, Pulleyblank B, Raghavan P, Sudan M. The minimum latency problem. In: Proceedings of the 26th annual ACM symposium on the theory of computing. Montreal, ACM; 1994. p. 163–71.
[6] Campbell AM, Vandenbussche D, Hermann W. Routing for relief efforts. Transportation Science 2008;42(2):127–45.
[7] Coene S, Spieksma FCR. Profit-based latency problems on the line. Operations Research Letters 2008;36(3):333–7.
[8] Dewilde T. Het profit-based latency probleem. Masters thesis, University of Leuven, Leuven, Belgium; 2009 [in Dutch].
[9] Erkut E, Zhang J. The maximum collection problem with time-dependent rewards. Naval Research Logistics 1996;43(5):749–63.
[10] Feillet D, Dejax P, Gendreau M. Traveling salesman problems with profits. Transportation Science 2005;39(2):188–205.
[11] Glover F, Kochenberger GA, editors. Handbook of metaheuristics. Massachusetts: Kluwer Academic Publishers; 2003.
[12] Goemans M, Kleinberg J. An improved approximation ratio for the minimum latency problem. Mathematical Programming 1998;82(1–2):111–24.
[13] Lucena A. Time-dependent traveling salesman problem – the delivery man case. Networks 1990;20(6):753–63.
[14] Melvin J, Keskinocak P, Koenig S, Tovey C, Ozkaya BY. Multi-robot routing with rewards and disjoint time windows. In: Proceedings of the IEEE international conference on intelligent robots and systems, San Diego; 2007. p. 2332–7.
[15] Mladenovic N, Hansen P. Variable neighbourhood search. Computers & Operations Research 1997;24(11):1097–100.
[16] Ngueveu SU, Prins C, Wolfler-Calvo R. An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Computers & Operations Research 2010;37(11):1877–85.
[17] Rosenkrantz DJ, Stearns RE, Lewis PM. An analysis of several heuristics for the traveling salesman problem. In: Ravi SS, Shukla SK, editors. Fundamental problems in computing. New York: Springer Science; 2009. p. 45–68.
[18] Salehipour A, Sörensen K, Goos P, Bräysy O. An efficient GRASP+VND metaheuristic for the traveling repairman problem. 4OR 2011;9(2):189–209.
[19] Talbi EG. Metaheuristics: from design to implementation. New Jersey, Wiley; 2009.
[20] Tang H, Miller-Hooks E, Tomastik R. Scheduling technicians for planned maintenance of geographically distributed equipment. Transportation Research Part E 2007;43:591–609.
[21] Vansteenwegen P, Souffriau W, Van Oudheusden D. The orienteering problem: a survey. European Journal of Operational Research 2011;209(1):1–10.
[22] Wu BY, Huang Z-N, Zhan F-J. Exact algorithms for the minimum latency problem. Information Processing Letters 2004;92(6):303–9.