Erasmus University Rotterdam

Erasmus School of Economics

Bachelor Thesis
Econometrics & Operations Research

A TABU Search Heuristic for the Capacitated Team Orienteering Problem

Author:
Mieke Jansen

Student number:
426021

Supervisor:
M.A. van Zon MSc

Second assessor:
Prof. dr. A.P.M. Wagelmans

July 8, 2018

Abstract

In this paper, we develop a heuristic algorithm to solve the Capacitated Team Orienteering Problem (CTOP), which is a capacitated version of the Team Orienteering Problem (TOP). In the TOP, a number of vehicle tours need to be constructed such that the total collected reward is maximized, while a limitation on the tour duration exists. In the CTOP, additionally, customers have a non-negative demand and vehicles face capacity constraints. Our solution technique builds upon the solution method for the TOP as proposed by Tang and Miller-Hooks (2005). An adaptive memory procedure incorporates a tabu search procedure, which makes use of variable neighbourhood search and executes random and greedy tour improvement procedures. Computational experiments reveal that our heuristic is able to yield high-quality solutions for the TOP as well as the CTOP in an efficient way.

Contents

1 Introduction

This paper presents and justifies the need to investigate a tabu search heuristic for the Capacitated Team Orienteering Problem. It provides methods that will be used to implement, extend and improve a tabu search approach for solving the Team Orienteering Problem (TOP) as proposed by Tang and Miller-Hooks (2005). In the TOP, the goal is to find a fixed number of vehicle tours such that profit received from customers is maximized. In doing so, a pre-specified time limit of a tour may not be exceeded. The TOP originates from the outdoor sports game called orienteering, where participants try to visit as many control points as possible. If a participant does not return to the finishing point in time, he or she is disqualified. The TOP can be regarded as the multi-competitor version of the orienteering game. However, it has many more practical applications. An example is the recruitment of college football players (Butt and Cavalier, 1994), where the recruiter cannot possibly visit all high schools in the region to recruit players for the college football team. To optimize the subset of high schools in the region to be visited, every high school is rewarded a positive weight based on historical performance of this high school. Taking into consideration school hours, a limitation exists on the available time per day to make visits. Regarding the high schools as vertices and the college campus where the recruiter needs to start the day and return to at the end of the day as a depot, this recruitment problem can be modelled as a TOP. Also, many pickup and delivery problems can be modelled as TOPs, where time limits of tours exist due to work day length restrictions.

The TOP is closely related to the Orienteering Problem (OP) and the Vehicle Routing Problem (VRP). The OP, also known as the Selective Traveling Salesman Problem (STSP), consists of constructing a single tour while maximizing profits and not exceeding a stipulated time limit. In the TOP, however, besides selecting and ordering customers in a tour, also an assignment decision needs to be made, on which customers should be assigned to which vehicles. Moreover, the TOP differs from the VRP in two important ways. Firstly, the sets of vertices to be covered are significantly different. Whereas in the VRP all customers need to be served in the set of constructed tours, the constructed tours in the TOP might include only a subset of the customers. Secondly, the objectives of the VRP and the TOP differ. Whereas in the VRP the goal is to minimize the cost associated with the constructed tours, the TOP strives to maximize the total profit associated with the visits to customers.

As a consequence of these differences, techniques to address the widely studied (S)TSP and VRP are not straightforwardly applicable to the TOP. Nevertheless, the TOP is a problem with many practical applications, as was pointed out earlier. Therefore, it is relevant to investigate solution methods for this problem.

The first study to propose an exact method to solve the TOP was presented by Butt and Ryan (1999). In their approach they combined column generation and branch-and-bound algorithms to solve a set-partitioning formulation of the TOP. A column generation technique was also used by Gueguen (1999), who also touched upon time windows for visiting customers. Following the study by Gueguen (1999), Boussier et al. (2007) presented an algorithm that solves the TOP exactly. Their method builds on Gueguen's Branch and Price algorithm, developing more refined techniques as well as a generic Branch and Price scheme that can be applied to different types of orienteering problems, amongst which the TOP. More recently, El-Hajj et al. (2016) presented a cutting planes algorithm that was able to find an optimal solution to instances that were not earlier solved to optimality.

Laporte and Martello (1990) proved that the OP is NP-hard. Since the OP is the single-vehicle version of the TOP, the TOP can also be concluded to be NP-hard. Therefore, in literature that considers solution methods for the TOP, the focus often lies on heuristic approaches. The aforementioned algorithms are able to solve small to medium instances of the TOP exactly. However, since the aim of this paper is to solve

relatively large problem instances of the TOP, a heuristic approach is preferred to exact solution methods.

Butt and Cavalier (1994) were one of the first to present a heuristic procedure for solving the TOP, which showed to yield high-quality solutions within a reasonable amount of time. Tsiligirides (1984) developed a stochastic algorithm to solve the one-vehicle variant of the TOP, earlier introduced as the OP. Chao et al. (1996b) modified this heuristic to make it applicable to the TOP and compared it with a self-constructed heuristic for the TOP. Results showed that the heuristic as constructed by Chao et al. (1996b) outperformed the modified heuristic by Tsiligirides (1984) in 60% of the test problems and that the latter produced better solutions in only 7% of all test problems. Before 2005, no study proposed to perform a tabu search method for solving the TOP. Since a tabu search approach appeared to provide close to optimal solutions for the OP in a study by Gendreau et al. (1998), Tang and Miller-Hooks (2005) proposed a tabu search heuristic for the TOP, which also turned out to produce solutions of high quality. Their results outperformed those of the heuristics as proposed by Chao et al. (1996b), whose 5-step meta-heuristic appeared to be the best published heuristic for the TOP until then. Archetti et al. (2007) proposed numerous variable neighbourhood search methods to solve the TOP. In particular, they showed that on average, the performance of all their algorithms is better than that of the tabu search method by Tang and Miller-Hooks (2005). Ke et al. (2008) used an ant colony optimization approach for the TOP and concluded that this approach belongs to one of the most efficient and effective solution approaches for the TOP thus far. Vansteenwegen et al. (2009) extended the existing studies on the TOP by introducing time windows. The incorporation of time windows might be valuable in case of impatient customers or opening hours of places to be visited. Not only did their iterated local search heuristic yield satisfying solutions, Vansteenwegen et al. (2009) also managed to reduce the computation time with a factor of several hundreds with respect to the best known solution.

As mentioned earlier, Gendreau et al. (1998) developed a tabu search heuristic for the OP that was able to provide solutions that approximated optimal solutions. Moreover, Tang and Miller-Hooks (2005) managed to construct an effective and efficient tabu search algorithm for the TOP. The flexible structure of their tabu search procedure allows for a wide applicability of the algorithm that they proposed. It is therefore useful to further investigate and refine this heuristic.

Tang and Miller-Hooks (2005) justify their choice to make use of tabu search by its good performance in the OP (Gendreau et al., 1998) and other complex combinatorial optimization problems in general (Glover, 1989, 1990). The strength of tabu search lies particularly in the ability to forbid a certain solution area, allowing the process to move out of a local optimum. The tabu search algorithm will be incorporated in an Adaptive Memory Procedure (AMP). Benefits of using an AMP to embed the tabu search algorithm include its ability to save time by taking into account earlier computations and to exhibit a number of different solutions of high quality (Taillard et al., 2001).

In this paper, first, parts of the research by Tang and Miller-Hooks (2005) will be replicated. This means that a tabu search algorithm will be implemented as part of an Adaptive Memory Procedure, and computational experiments will be re-executed. Subsequently, an extension to this heuristic will be suggested and implemented, in order to be able to solve not only the TOP but also the Capacitated TOP (CTOP). In this special case of the TOP, customers have a non-negative demand, and we do not only face tour duration constraints, but also vehicle capacity constraints. A frequently encountered application of the CTOP is the delivery of packages by a fleet of vehicles. Besides work day length constraints, vehicles will not be able to carry an unlimited number of packages. Thus far, little research has been conducted to the CTOP. Archetti et al. (2009) developed meta-heuristics to solve the CTOP that were based on tabu search in combination with variable neighbourhood search. Tarantilis et al. (2013) incorporated a variable neighbourhood search method

in a bi-level framework, leading to results comparable to those of Archetti et al. (2009). The methods described our this paper will be applied to problem instances used by Archetti et al. (2009) and we will compare our results to theirs.

The remainder of this paper is structured as follows. In Section 2, a mathematical formulation of the TOP is provided. In Section 3, the Adaptive Memory Procedure and the incorporated tabu search procedure will be described thoroughly. Section 4 provides results of computational experiments conducted on data instances for the TOP and the CTOP. These results will be compared with those obtained by Tang and Miller-Hooks (2005) and Archetti et al. (2009). Finally, in Section 5, conclusions will be drawn.

2 Mathematical formulation

The CTOP as described in Section 1 can be formulated as an integer program. This section provides mathematical notation of this integer program, which helps to understand the structure of the CTOP. To be able to provide a mathematical formulation of the CTOP, we introduce some notation. The CTOP can be considered as a complete graph $G = (V, E)$, where $V = \{0, 1, \ldots, n-1\}$ and $E = \{(i, j) \mid i, j \in V\}$ represent the vertex set and the edge set, respectively. The goal is to construct m vehicle tours which lead to the collection of maximum reward. Each vehicle has an identical maximum tour duration L and maximum capacity Q. In V, both vertices 0 and $n-1$ represent depots, being the starting point and finishing point of each tour, respectively, and we define $D = \{0,\ n-1\}$ as the set of depots. For the sake of clear notation, starting depot 0 will be denoted as D_s and finishing depot $n-1$ will be denoted as D_f. Consequently, $V \backslash D$ represents the set of vertices in the graph that are not a depot. The vertices in this set will be referred to as customer vertices. Visiting a customer vertex always comes along with a positive reward. A depot is always associated with zero reward. The starting depot and finishing depot of a vehicle tour might differ from each other, but both are equal for each vehicle in the CTOP.

Four parameters are considered. d_{ij} denotes the travel distance from vertex i to vertex j. This distance is computed as the Euclidian distance between two vertices. As a result of using the Euclidian distance as a distance measure, the triangle inequality holds; visiting an extra vertex in a route will always result in an increase of the tour duration. Three parameters are associated with each customer vertex. s_i denotes the service time at vertex i, which, together with d_{ij}, determines the tour duration. Visiting vertex i results in the collection of reward r_i. The demand of customer vertex i is denoted by q_i, which is an essential parameter regarding the limited capacity of each vehicle.

We introduce two decision variables, one associated with edges and one associated with vertices. x_{ijk} represents the number of times that edge (i, j) is traversed by vehicle k. The number of times that vertex i is visited by vehicle k is denoted by y_{ik}. The fact that every customer vertex can be visited at most once implies that both decision variables are binary variables.

The CTOP can now be formulated as follows:

$$\max \quad \sum_{i=1}^{n-2}\sum_{k=1}^{m} r_i y_{ik}, \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n-2}\sum_{k=1}^{m} x_{D_s jk} = m, \tag{2}$$

$$\sum_{i=1}^{n-2}\sum_{k=1}^{m} x_{iD_f k} = m, \tag{3}$$

$$\sum_{i<j} x_{ijk} + \sum_{i>j} x_{jik} = 2y_{jk} \qquad j = 1, 2, \ldots n-2;\ k = 1, 2, \ldots, m, \tag{4}$$

$$\sum_{i=0}^{n-3}\sum_{j>i} d_{ij}x_{ijk} + \sum_{i=1}^{n-2} s_i y_{ik} \leq L \qquad k = 1, 2, \ldots, m, \tag{5}$$

$$\sum_{i=1}^{n-2} q_i y_{ik} \leq Q \qquad k = 1, 2, \ldots, m, \tag{6}$$

$$\sum_{k=1}^{m} y_{ik} \leq 1 \qquad i = 1, 2, \ldots, n-2, \tag{7}$$

$$\sum_{i,j\in U, i<j} x_{ijk} \leq |U| - 1 \qquad U \subset V\backslash D;\ 2 \leq |U| \leq n-3;\ k = 1, 2, \ldots, m, \tag{8}$$

$$x_{ijk} \in \{0, 1\} \qquad 0 \leq i < j \leq n-1;\ k = 1, 2, \ldots m, \tag{9}$$

$$y_{ik} \in \{0, 1\} \qquad i = 1, 2, \ldots n-2;\ k = 1, 2, \ldots m. \tag{10}$$

The objective function (1) represents the total reward that is collected by all m vehicles. Constraints (2) and (3) ensure that exactly m tours start in the starting depot D_s and finish in the finishing depot D_f. Constraints (4) assure that when a vehicle visits a customer vertex, it must enter this vertex through an edge and leave it through another. In this way, connectivity of each tour is guaranteed. Constraints (5) prohibit each tour from exceeding the preset time limit. Since service time of customers is not taken into consideration in this paper, constraint (5) can be simplified to

$$\sum_{i=0}^{n-2}\sum_{j>i} d_{ij}x_{ijk} \leq L. \tag{11}$$

Constraints (6) impose a restriction on the used capacity of each vehicle. Constraints (7) ensure that each customer vertex is visited by at most one vehicle. The existence of sub-tours is prevented by (8). Lastly, (9) and (10) assure integrality of the decision variables.

Note that by setting the demand of each customer to zero, i.e., $q_i = 0\ \forall i \in V\backslash D$, and by setting the capacity Q to any strictly positive number, the CTOP can be generalized to the TOP.

3 Methodology

Tabu search comes down to a local neighbourhood search where certain solutions are prohibited. In this way, worsening of a solution is allowed in the hope that the process can steer out of a local optimum. In this paper, the tabu criterion constitutes removing a vertex from a solution tour which was only recently inserted. 'Recently' is a broad term, but is specified by the tabu parameter θ, which will be elaborated on in Section 3.2.1. A solution is a tabu solution when this criterion was violated during the searching process. The tabu search algorithm will always return a non-tabu neighbour, unless a found tabu solution comes along with a higher reward than any solution found thus far. A neighbourhood solution, or neighbour, is defined as a solution which contains the same number of vehicle tours as the initial solution and in which at least one vertex is visited that was not contained in any of the vehicle tours in the initial solution.

During the tabu search procedure, a method of variable neighbourhood search is employed. This means that during the searching process, the solution space that is explored can vary. In particular, we will switch between small and large neighbourhoods exploration stages during the searching process. When investigating a large exploration neighbourhood yields an improved solution over the current best feasible solution, the searching process will switch to a small neighbourhood stage, granting the process to explore a smaller scope of possible solutions in the next iteration. If investigating a small exploration neighbourhood yields dissatisfying results (i.e. no improved solutions are being found within a given time frame), the searching process will return to the large neighbourhood exploration stage. Although exploring a large neighbourhood might lead to better neighbourhood solutions, we need to take into account computational efficiency of our algorithm, which is why the tabu search process will also return to small neighbourhood stages. This variable neighbourhood search is based on the approach of Tang and Miller-Hooks (2005), whose empirical experiments showed that alternation between neighbourhood sizes indeed leads to an efficient searching process, without major degradation in solution quality.

The tabu search algorithm as developed in Tang and Miller-Hooks (2005) is incorporated in an Adaptive Memory Procedure (AMP). In an AMP, a memory is created in which partial solutions are stored. In our application of the CTOP, a partial solution is defined as one single vehicle tour. In other words, a solution to the CTOP always contains m partial solutions. As multiple iterations of the AMP are executed, this memory is updated by storing partial solutions that were constructed by a local search procedure; hence the name 'adaptive memory' procedure. The updated memory will always be used as the starting point of a new AMP iteration. In general, the AMP encompasses the following steps (Olivera and Viera, 2007):

Step 1. Initialize the adaptive memory,

Step 2. While a stopping criterion is not met:

   a. Construct a new initial solution S by combining partial solutions from the adaptive memory,

   b. Perform local search in order to improve the initial solution S. S^* is the improved solution,

   c. Update the adaptive memory by replacing low-quality partial solutions by higher-quality partial solutions from S^*,

Step 3. Output a solution by combining partial solution from the adaptive memory.

In the AMP that is described in this paper, Step 1 can be identified as the generation of vehicle tours and storing these in the adaptive memory. In Step 2a, m vehicle tours are combined to construct an initial solution. The local search method that will be used in Step 2b is a tabu search procedure. The tabu

search procedure provides a neighbourhood solution to the initial solution, whose tours replace tours that are currently stored in the AMP, if they come along with a higher reward, in Step 2c. This means that in every AMP iteration, at most m vehicle tours in the AMP are replaced. Step 2 will be repeated a preset number of times, after which a solution is outputted in Step 3. This step will encompass solving a variant to the set packing model in order to be able to output the best possible solution.

The remainder of this section provides an extensive description of the steps to be executed in the AMP and the tabu search procedure in Section 3.1 and 3.2, respectively.

3.1 Adaptive Memory Procedure (AMP)

The AMP that is described in this paper starts with generating a set of single vehicle tours. All generated vehicle tours are stored in an adaptive memory. From these tours in the adaptive memory, a combination of m tours is selected as an initial solution.

This initial solution is improved by using it as an input for the tabu search algorithm. In the tabu search procedure, a number of neighbourhood solutions to the initial solution is constructed. The best non-tabu neighbour is selected to be outputted, unless a tabu neighbour yield higher reward than the current best feasible solution. After the tabu search procedure has been performed, the adaptive memory is updated by replacing existing vehicle tours by vehicle tours that were created in the tabu procedure. A global overview of the steps in the AMP and in the tabu search procedure is provided below.

Adaptive Memory Procedure

1. Generate N vehicle tours.
2. Construct an initial TOP solution by combining vehicle tours in the adaptive memory.
3. Improve initial solution by tabu procedure.
   A. Initialize tabu parameters and neighbourhood size.
   B. Generate neighbourhood solutions to the initial solution.
   C. Evaluate all generated neighbours and determine which neighbour is of highest quality.
4. Update the adaptive memory.

During the execution of the AMP, several parameters are used, whose definitions can be found in Table 1. The values of these parameters are fixed during the entire AMP. High values of N and λ are likely to lead to solutions of higher quality. However, their impact on computational effort that is required to execute the entire AMP needs to be taken into consideration when deciding on their values.

Table 1: *AMP parameters*
This table provides the definition of the parameters used in the Adaptive Memory Procedure and the steps in which they occur.

Parameter	Definition	Step
λ	Number of times that the AMP is executed before a final solution is outputted.	4
N	Number of vehicle tours that are stored in the Adaptive Memory.	1
T	Number of potential vehicle tours to be selected when constructing a feasible initial solution.	2

3.1.1 Step 1: Generation of N vehicle tours

The first step of the AMP comes down to the repeated execution of a procedure in which m feasible vehicle tours are created. A feasible tour is defined as one of which the duration does not exceed time limit L. The duration of a tour of vehicle k is denoted by $D(t_k)$.

In the construction of a vehicle tour, decisions need to be made on which vertices to include in a tour. Given that the starting vertex and finishing vertex of each vehicle tour are known, constructing a tour comes down to selecting a subset of customer vertices and deciding on the order in which they are visited. In this decision process, an evaluation function is employed in order to valuate the quality of a vertex insertion into a tour. The evaluation function that is employed when considering the insertion of a vertex j between vertices p and q in a tour takes into account the ratio of the extra distance and the extra reward that this insertion brings about, as well as the relative required capacity needed to insert this vertex. In particular,

$$E(j,p,q) = \frac{d_{pj} + d_{jq} - d_{pq}}{r_j} * \frac{q_j}{C}. \tag{12}$$

In the procedure that follows, an insertion is regarded of higher quality if the evaluation function in (12) is lower. Since the space in the adaptive memory is specified at N, this procedure is executed $\lceil N/m \rceil$ times in order to guarantee that at least N vehicle tours are created:

Creating m vehicle tours

1. Create m 'dummy' tours $t_k = \{D_s, D_f\}$. Let SE be the set of vertices that can be added to a dummy tour without violating the time limit constraint:

   $$SE = \{j \mid d_{D_s j} + d_{j D_f} \leq L, \forall j \in V\}.$$

   Set $k = 1$.

2. If t_k contains no vertices but the depots, select a random vertex $j \in SE$ and find two vertices p and q in t_k for which $E(j,p,q)$ is minimal. Otherwise, select $j \in SE$ and two vertices p and q for which $E(j,p,q)$ is minimal **and** for which the updated duration of t_k does not exceed time limit L.
   If no such j can be found, go to Step 3. Otherwise, insert j in t_k between p and q, set $SE = SE \backslash \{j\}$ and repeat Step 2.

3. Set $k = k + 1$. If $k > m$, stop. Otherwise, go to Step 2.

The N vehicle tours that have been created are stored in the adaptive memory. Furthermore, the iteration counters q_{AMP} and q_{tabu} are initialized at 0.

3.1.2 Step 2: Construction of an initial solution

In the second step of the AMP, the N tours that were generated in Step 1 are assigned a probability to be selected in the initial solution. This probability is based upon the objective value of a tour, where tours with a high reward will have a higher probability to be selected than those with low rewards. A more extensive description of this procedure is provided below, where $R(t_k)$ denotes the total reward that is collected by vehicle tour t_k.

Constructing a feasible initial solution

1. Let candidate list CL contain all N tours in the adaptive memory. Select T vehicle tours with highest rewards from the adaptive memory and sort them in a non-increasing order. Let $Index(0) = 0$ and assign each tour $t_i, i = 1, \ldots, T$ an index value:

$$Index(i) = \frac{\sum_{j=1}^{i} R(t_j)}{\sum_{j=1}^{T} R(t_j)}. \tag{13}$$

2. Draw a random number from the uniform distribution on the interval $[0, 1]$. Select tour t_i for which the random number lies between $Index(i-1)$ and $Index(i)$. Add tour t_i to the current solution and remove all tours from CL that have at least one customer vertex in common with t_i.

3. If $CL = \emptyset$ and the current solution contains less than m tours, go to Step 4. If the current solution contains m tours, stop. Otherwise, go to Step 2.

4. Execute the procedure *Creating m vehicle tours* in Section 3.1.1: every time a tour is created, add it to the current solution. If the current solution contains m vehicle tours, stop.

3.1.3 Step 3: Improvement of the initial solution

In this step, the tabu search procedure is called in an attempt to improve the initial feasible solution. The tabu procedure can be characterized into three main steps: (A) Initialization, (B) Improvement and (C) Evaluation. However, before the tabu procedure can be performed on the feasible initial solution, the iteration counter q_{stepC} needs to be initialized at 0. This iteration counter is used to keep track of the number of times that Step C of the tabu procedure is performed. This number is restricted to a maximum of μ, after which the tabu procedure will terminate and a solution will be outputted.

In Section 3.2, the tabu procedure will be explained in detail. The procedure takes the initial feasible solution as input and returns a solution with m vehicle tours, which will be used as an input for Step 4 of the AMP. The tabu procedure employs both random and greedy procedures in the searching process. These methods are not only aimed at maximizing collected reward, but also at minimizing the travelled distance of each vehicle. Shorter tours can facilitate the insertion of extra vertices in successive iterations, which can ultimately lead to higher reward of the final solution.

3.1.4 Step 4: Update of the adaptive memory

In the final step of the AMP, the adaptive memory is updated and eventually, if the AMP has been executed λ times, a final solution is outputted. Otherwise, the AMP returns to Step 2 and uses the updated adaptive memory to construct a new solution.

When updating the adaptive memory, we attempt to replace low-reward vehicle tours by vehicle tours that yield higher reward. For each tour t_k in solution S that was outputted in Step 3 of the AMP, we check whether its reward is higher than that of tour t_{low}, the tour with lowest reward in the adaptive memory. If this is true, t_{low} is removed from the adaptive memory and replaced by t_k.

The AMP-counter is updated, that is, $q_{AMP} = q_{AMP} + 1$. If $q_{AMP} < \lambda$, the procedure returns to Step 2 of the AMP. Otherwise, the AMP terminates by identifying and outputting the best solution. To output the best solution, two different approaches are followed. Firstly, every solution that was ever outputted by the tabu procedure and passed to Step 4 of the AMP is stored. The solution with the highest reward is selected as the best solution. Call this solution S^*_{tabu}.

The second approach to construct a solution is to combine m tours that are currently stored in the adaptive memory. To find an optimal combination of tours, a variant of the set packing problem is solved exactly. To do so, we introduce the binary decision variable x_i, which is equal to 1 if tour i is selected in the final solution and 0 otherwise. Additionally, two parameters are used. a_{ij} is equal to 1 if tour i visits customer vertex j. The total reward that is collected when tour i is executed is denoted by r_i.

The variant of the set packing problem that will be solved can be modelled as follows:

$$\max \sum_{i=1}^{N} r_i x_i, \tag{14}$$

$$\text{s.t.} \sum_{i=1}^{N} a_{ij} x_i \leq 1 \qquad j \in V \backslash D, \tag{15}$$

$$\sum_{i=1}^{N} x_i \leq m, \tag{16}$$

$$x_i \in \{0, 1\} \qquad i = 1, \ldots, N. \tag{17}$$

In this formulation, the objective is to maximize the total reward collected from executing the tours that are selected in the solution, as shown in (14). Constraints (15) prohibit a solution from containing tours with a customer node in common. In other words, a tour can be added to the solution if and only if it visits solely customer nodes which were not yet visited by other tours in the current solution. Constraint (16) ensures that the number of selected tours does not exceed the number of available vehicles m. Solving this set packing problem yields the optimal combination of tours contained in the adaptive memory. Call this solution S^*_{AM}.

The final solution that is outputted by Step 4 of the AMP is determined as follows:

$$S^* = \underset{S \in \{S^*_{tabu}, S^*_{AM}\}}{\arg\min} R(S). \tag{18}$$

Note that the total reward of a solution S is denoted by $R(S)$, not to be confused with $R(t_k)$ as used in Equation (13), which represents the total reward of a *partial* solution, i.e. the total reward of a single vehicle tour.

3.2 Tabu search procedure

The tabu search procedure that is called from the AMP can be divided into three main steps: initialization, solution improvement and evaluation. In the initialization, a set of tabu parameters is determined. This set depends on the solution space that will be explored in this step. Recall that during the search process, Tang and Miller-Hooks (2005) decided to make use of variable neighbourhood search: the method alternates between small and large stages of neighbourhood exploration, as was explained earlier in Section 3. In the initialization step, the tabu parameters are set to a small neighbourhood stage.

In the second step, random and greedy procedures are applied to explore different neighbourhoods and to generate corresponding neighbourhood solutions. These solutions are improved by balancing longer and shorter tours and by exchanging vertices between two tours. When neighbour solutions have been generated, the evaluation step selects a solution from all generated solutions which yields the highest reward and is not on the tabu list. During the tabu search procedure, we will always keep track of a list of vertices that cannot be removed from a tour in the current iteration. If a vertex from this list is removed anyways, the tabu status of the solution is updated. A solution is put on the tabu list when, during the searching process, a vertex is removed from a tour less than θ iterations after it was inserted in this tour. θ is a random number that depends on the two tabu parameters θ_{min} and θ_{max}, which are initialized in the first step of the tabu procedure. A solution that is on the tabu list can only be selected if its reward is higher than the reward of the best found solution thus far. The solution that is selected in the evaluation step will be used as the new initial solution for the next tabu iteration. The following sections will give a comprehensive description of each step in the tabu search procedure.

3.2.1 Step A: Initialization

In Step A of the tabu procedure, the tabu parameters are initialized to explore a small neighbourhood stage. This means that a relatively small number of neighbourhood solutions of the initial solution from the AMP will be be generated in Step B. The tabu parameters are defined as follows:

Table 2: *Tabu parameters*
This table gives the definition of the parameters in the tabu procedure and the steps in which they occur.

Parameter	Definition	Step
α	Maximum number of non-improvement iterations in the tabu search procedure.	C.3
β	Number of neighbourhood solutions that will be generated in the tabu search procedure.	B.6
δ	Every δ tabu iterations, the previously selected non-tabu solutions will be examined. If all of them are feasible, penalty coefficient η will be halved; if all of them are infeasible, η will be doubled.	C.3
ϵ	Tour duration limit L is sometimes adjusted to $L \cdot (1 + \epsilon)$ in the tabu search procedure.	B.3
ϕ	Maximum capacity Q is sometimes adjusted to $Q \cdot (1 + \phi)$ in the tabu search procedure.	B.3
η	Penalty coefficient applied to the rewards of solutions that were generated in the tabu search procedure, which are infeasible due to time limit constraint.	C.1
ζ	Penalty coefficient applied to the rewards of solutions that were generated in the tabu search procedure, which are infeasible due to capacity constraint.	C.1
μ	Maximum number of times that Step C of the tabu procedure is executed.	C.3
$[\theta_{min}, \theta_{max}]$	Tabu tenure θ for each recently inserted vertex is randomly chosen from the interval $[\theta_{min}, \theta_{max}]$.	B.3,B.4, B.5

3.2.2 Step B: Generation and improvement of neighbourhood solutions

In Step B of the tabu procedure, a number of neighbourhood solutions of the current solution are generated and improved by using heuristic procedures. The number of generated neighbourhood solutions is equal to the parameter β, which varies with the size of the neighbourhood. Recall that a neighbour is defined as a combination of m tours that contains at least one vertex which is not contained in the initial solution. Generating and improving a neighbour encompasses at most five steps, which are repeated β times. This results in the generation of β neighbours to the initial solution.

Besides keeping track of the tabu status of a solution, the feasibility of every generated solution is stored. This is necessary in order to determine whether penalty coefficients η and ζ can be halved or need to be doubled in the evaluation step. Infeasible solutions can occur when a solution tour exceeds the stipulated time limit or maximum capacity. Allowing this to happen during the tabu procedure can help the searching process to escape from a local minimum. Therefore, we will see that in Step B.3, B.4 and B.5, solutions might turn out infeasible. However, infeasible solutions as final solutions are not desirable and will thus be penalized when evaluating their quality.

Step B of the tabu search procedure encompasses a number of sub-steps, which can be summarized as follows:

Tabu procedure Step B - Overview

0. Let neighbourhood counter $p = 0$, set the current initial solution to S. Let $\bar{S}$ be the set of vertices that are not contained in the tours of S.
1. Randomly select two distinct tours t_1 and t_2 from S.
2. Try to insert additional vertices in t_1 and t_2. If any vertex could be inserted into either t_1 or t_2, go to Step 4.
3. Exchange vertices between t_1 and t_2 and $\bar{S}$.
4. Try to balance between t_1 and t_2 by removing a node from the longer tour and inserting it in the shorter tour. If this was successful, go to Step 6.
5. Exchange vertices between t_1 and t_2 in order to decrease the total duration.
6. Let $p = p + 1$ and $q_{tabu} = q_{tabu} + 1$. If $p < \beta$, go to Step 1.

Depending on whether a preceding step was successful, not every sub-step will always be executed. In the remainder of this section, a more thorough description of steps B.1-B.5 will be provided. In Appendix A, a visual representation of the solution tours during different stages of Step B of the tabu procedure is provided for two TOP problem instances.

Step B.1 In this step, two distinct tours are randomly chosen from the initial solution S that was passed from Step 2 of the AMP. In the remainder of this section, these two tours will be referred to as t_1 and t_2.

Step B.2 For each tour t_1 and t_2, a variant of the cheapest insertion procedure for the Travelling Salesman Problem is executed. In this procedure, the evaluation function that is used takes into account solely the extra distance associated with the insertion of vertex j in between vertices p and q, that is:

$$E(j,p,q) = d_{pj} + d_{jq} - d_{pq}. \tag{19}$$

Insertion procedure

1. Let SE be the set of vertices in $\bar{S}$ that can be inserted in a dummy tour without violating the time limit constraint and capacity constraint:

   $$SE = \{j \mid d_{D_s j} + d_{j D_f} \leq L, q_j \leq Q, \forall j \in \bar{S}\}.$$

2. Select $j \in SE$ and two vertices p and q in tour t_k for which $E(j,p,q)$ is minimal **and** for which the updated duration of t_k does not exceed time limit L **and** for which the updated used capacity does not exceed vehicle capacity Q.
   If no such j can be found, continue to Step 3. Otherwise, insert j in t_k between p and q, set $SE = SE \backslash \{j\}$ and repeat Step 2.

3. If Step 2 was successful, go to Step B.4 of the tabu procedure. Otherwise, go to Step B.3 of the tabu procedure.

Step B.3 If in Step B.2 no vertices could be inserted into any of the tours t_1 and t_2, the tabu search procedure continues to step B.3. In this step, for each tour t_1 and t_2, vertices are exchanged with $\bar{S}$.

Exchanging vertices between t_k and $\bar{S}$

1. Let $numRemove$ be a number that was randomly drawn from the uniform distribution on the interval between 0 and the number of customer vertices in t_k. Set $q_{removed} = 0$.

2. Select a random customer vertex in t_k and remove it from the tour. Let $q_{removed} = q_{removed} + 1$.

3. If $q_{removed} < numRemove$, repeat Step 2. Otherwise, go to Step 4.

4. Select a random vertex $j \in \bar{S}$. Find the position in t_k that entails the smallest increase in tour duration d_{extra}. This location is found by applying the evaluation function as given in Equation (19). If $D(t_k) + d_{extra} < L(1+\epsilon)$ and $C(t_k) + q_j < Q(1+\phi)$, insert j at this position in t_k.

5. If $D(t_k) < L$, $C(t_k) < Q$, $\bar{S} \neq \emptyset$ and not all $j \in \bar{S}$ have been checked yet, repeat Step 4. Otherwise, stop.

Step B.4 In the fourth step of the tabu search procedure, an attempt is made to move a vertex from the longer to the shorter of the two tours t_1 and t_2. This might result in a better balance between the durations of the two tours.

> **Balancing longer and shorter tour**
>
> 1. Let $t_{long,old}$ be the tour with the longest duration and $t_{short,old}$ the one with the shortest duration.
> 2. For each customer vertex in $t_{long,old}$, investigate whether it is possible to insert this vertex into $t_{short,old}$, resulting in $t_{long,new}$ and $t_{short,new}$, such that $D(t_{long,new})+D(t_{short,new}) < D(t_{long,old})+ D(t_{short,old})$.
> 3. If Step 2 was successful, proceed to Step B.6 of the tabu procedure. Otherwise, go to Step B.5 of the tabu procedure.

Note that in the second step of this procedure, the solution might become infeasible. This can occur when the duration of the tour that used to be the short tour, exceeds time limit L after the insertion of the vertex from $t_{long,old}$. The condition of total tour duration decrease, i.e. $D(t_{long,new}) + D(t_{short,new}) < D(t_{long,old}) + D(t_{short,old})$, might still be satisfied in this case.

Step B.5 If the replacement of a vertex from the shorter to the longer tour in Step B.4 was not successful, Step B.5 will make a one-vertex exchange between t_1 and t_2.

> **Exchanging vertices between t_1 and t_2**
>
> 1. Set $i = 1$.
> 2. Select the i^{th} customer vertex in t_1 and call this vertex x. For each customer vertex $y \in t_2$, investigate whether it is possible to exchange vertices x and y such that $D(t_{1,new}) + D(t_{2,new}) < D(t_{1,old}) + D(t_{2,old})$. If so, immediately make this exchange and proceed to Step 3.
> 3. Set $i = i + 1$. If $i > size(t_1)$, stop. Otherwise, repeat Step 2.

Just like in Step B.4, Step B.5 might also cause infeasibility of a solution. The same reasoning can be followed as for Step B.4; since the time limit capacity is not restrained, it might be violated even though the total duration of tours t_1 and t_2 has decreased.

3.2.3 Step C: Evaluation

In Step C of the tabu procedure, the best solution will be selected from the neighbours created in Step B.

Step C.1 In the selection process, infeasible solutions will be penalized by means of a reduction of their reward, leading to a 'quasi-reward' that is lower than the original reward. For feasible solutions, the quasi-reward will be equal to their original reward. The severeness of the penalty for infeasible solutions is determined by tabu parameter η. In particular, quasi-reward of any solution S is calculated by using the following

formula:

$$R^{quasi}(S) = \sum_{k=1}^{m} R(t_k) - \eta \sum_{k=1}^{m} \max(D(t_k) - L, 0) - \zeta \sum_{k=1}^{m} \max(C(t_k) - C, 0), \tag{20}$$

where the total reward of a tour t_k is denoted by $R(t_k)$ and the total used capacity of vehicle k is denoted by $C(t_k)$.

Step C.2 The best non-tabu and best tabu neighbour are denoted by S_{nt} and S_t, respectively. Recall that solution is considered tabu if a vertex that was inserted in a tour, was deleted from that tour within θ iterations after its insertion. θ is a random number that is chosen on interval $[\theta_{min}, \theta_{max}]$. Both S_{nt} and S_t are selected based on their quasi-rewards. Two cases are considered:

Selecting the best neighbour

- If the reward associated with the best tabu solution is higher than any solution found thus far, its tabu status can be overridden and this solution is selected as the best neighbour. The tabu-iteration counter is reset.
  $\rightarrow$ If $R(S_t) > R(S_{nt})$ and $R(S_t) > R(S^*)$, let $S^* = S_t, S = S_t$ and $q_{tabu} = 0$,
- In all other cases, the best non-tabu solution will be returned as best neighbour. If the reward of this non-tabu solution is higher than the best found solution thus far, S^* is updated and the tabu-iteration counter is reset.
  $\rightarrow$ Let $S = S_{nt}$. If $S_{nt} > S^*$, set $S^* = S_{nt}$ and $q_{tabu} = 0$.

Step C.3 In the last step, the best neighbour S is checked for feasibility and the current best feasible solution S_f^* is updated accordingly. Depending on the number of iterations that were needed for a solution improvement and on the feasibility of the neighbours that were generated in the last δ iterations, the neighbourhood size and tabu parameters α, η and ζ are updated.

Step C always starts with updating the step counter: $q_{stepC} = q_{stepC} + 1$. This counter is initialized at 0 in the AMP before the tabu procedure is called. If $q_{stepC} > \mu$, stop; proceed to Step 4 of the tabu procedure. Otherwise, execute the following procedure:

Check feasibility of S and update tabu parameters

- If S is feasible and $R(s) > R(S_f^*)$, let $S_f^* = S$ and $q_{tabu} = 0$,
- If at least one of the solutions found in the last δ tabu iterations was feasible, the penalty coefficients for non-feasible solutions can be halved, that is, $\eta = \eta/2$ and $\zeta = \zeta/2$,
- If all solutions found in the last δ tabu iterations were infeasible, the penalty coefficient for non-feasible solutions is doubled, that is, $\eta = 2\eta$ and $\zeta = 2\zeta$,
- If the neighbourhood that was explored in the last tabu iteration was small and $q_{tabu} > \alpha/2$, let the neighbourhood size in the next tabu iteration be large and go to Step B of the tabu procedure,

- If the neighbourhood that was explored in the last tabu iteration was small and $q_{tabu} \leq \alpha/2$, go to Step B of the tabu procedure,
- If the neighbourhood that was explored in the last tabu iteration was large and $q_{tabu} = 0$, let the neighbourhood size in the next tabu iteration be small and go to Step B of the tabu procedure,
- If the neighbourhood that was explored in the last tabu iteration was large and $0 < q_{tabu} \leq \alpha$, go to Step B of the tabu procedure,
- Otherwise, go to Step A of the tabu procedure.

4 Computational results

In this section, results of computational experiments will be presented and compared with the results found by Tang and Miller-Hooks (2005) and Archetti et al. (2009). The complete algorithm as described in this paper was implemented in Java. CPLEX (2016) was used to solve the set packing problem in Step 4 of the AMP exactly. All computations were executed on an Intel Core i5-4210U CPU 1.70GHz and 8GB RAM.

The data sets for the TOP that were used to test the tabu search algorithm in combination with an AMP originate from earlier literature (Chao et al., 1996b,a, 1993; Tsiligirides, 1984). For the CTOP, ten test instances were taken from Christofides et al. (1979). These instances originate from the VRP, meaning that travelling costs, customer demands and capacity constraints were present. Also, time constraints were already incorporated in these instances. No customer rewards were provided, which is why we used rewards as proposed by Archetti et al. (2009), who defined $r_i = (0.5 + h)q_i$, where h is a random number that was drawn from the uniform distribution on the interval $[0, 1]$. These 10 instances for the CTOP will be referred to as the benchmark instances.

An overview of the instances for the TOP and CTOP is provided in Table 3. Regarding the TOP, every data set contains multiple problem instances, each with the same set of vertices but differentiated by the values of m and L. In total, there are six data sets which contain 320 test problems in total. The reward that is associated with each customer vertex is random for data sets 1, 3, 4 and 7. For data sets 5 and 6, D_s and D_f are equal and rewards are proportional to the distance from this depot.

Table 3: *Summary of Data Sets*
This table shows (1) the number of vertices in each data set and (2) the shape of the geographical locations of these vertices.

(a) Instances for the TOP

Data Set	n	Shape
1	32	Random
3	33	Random
4	100	Random
5	66	Square
6	64	Diamond
7	102	Random

(b) Instances for the CTOP

Instance	n	m	Shape
3	101	15	Random
6	51	10	Random
7	76	20	Random
8	101	15	Random
9	151	10	Random
10	200	20	Random
13	121	15	Random
14	101	10	Random
15	151	15	Random
16	200	15	Random

The set of employed tabu parameters varies with the size of the neighbourhood. For data sets 1, 3, 4 and 7, the parameter sets are defined as follows:

$$\text{Small neighbourhood: } (\alpha, \beta, \delta, \epsilon, \phi, \eta, \mu, \theta_{min}, \theta_{max}) = (n, 2n, 6, 0.02, 0.02, 1, 10, 5, 10),$$

$$\text{Large neighbourhood: } (\alpha, \beta, \delta, \epsilon, \phi, \eta, \mu, \theta_{min}, \theta_{max}) = (n, 8n, 6, 0.05, 0.05, 1, 10, 6, 12).$$

Note that the values of penalty coefficients ϵ and ϕ are set equal in our computational experiments. Depending on how much value is assigned to the tour duration constraints and the capacity constraints, ϵ and ϕ can be modified accordingly. For data sets 5 and 6, different error terms ϵ and ϕ are employed than for the other data sets. For these two data sets, the time limit may not be exceeded in Step B.3 of the tabu procedure, and thus both values are set at zero. The remaining tabu parameters remain unchanged, resulting in the following parameter sets for data sets 5 and 6:

$$\text{Small neighbourhood: } (\alpha, \beta, \delta, \epsilon, \phi, \eta, \mu, \theta_{min}, \theta_{max}) = (n, 2n, 6, 0, 0, 1, 10, 5, 10),$$

$$\text{Large neighbourhood: } (\alpha, \beta, \delta, \epsilon, \phi, \eta, \mu, \theta_{min}, \theta_{max}) = (n, 8n, 6, 0, 0, 1, 10, 6, 12).$$

Parameters that are used in the AMP are $\lambda = 3$, $N = 1500$ and $T = 30$.

The AMP and tabu search algorithm as described in this paper were executed on each of the 320 TOP instances and 10 CTOP benchmark instances. Please note that since a heuristic approach is used for solving the NP-hard (C)TOP, the problems will not always be solved to optimality. Reported rewards always represent the best lower bound that our heuristic was able to find. When reporting the results, our results will be reported under 'Tabu'. Results of the tabu search heuristic by Tang and Miller-Hooks (2005) and Archetti et al. (2009) are referred to as 'TMH' and 'AFHS', respectively. Average rewards will always be rounded to the nearest integer. Additionally, (maximum) CPU times in seconds are always reported.

4.1 TOP results

The problem instances of the TOP were divided into 18 sub-categories, for which the average collected reward of both the implementation described in this paper and the tabu search heuristic by Tang and Miller-Hooks

(2005) is reported in Table 4. Also, the maximum CPU times in seconds is provided, determined based upon all problems in a problem category. Since in Step 4 of the AMP an integer programming problem is solved exactly, the algorithm has become more time-consuming. To compensate for this extra computation time, the Random Vertex Insertion (RVI) procedure that was used by Tang and Miller-Hooks (2005) was not incorporated in the algorithm as described in this paper. Testing the algorithm with a variant of this RVI procedure yielded minor or no improvements in the quality of the found solutions. This is confirmed by the results in Table 4, where the average reward of each problem category always lies within a 4.2% range from the average rewards found by Tang and Miller-Hooks (2005). We find that the solutions approximate the results by Tang and Miller-Hooks (2005) particularly well for data sets 1, 3, 5 and 6. This suggests that for large instances, incorporating a Random Vertex Insertion procedure might be beneficial. However, a trade-off will always need to be made between higher solution quality and extra computation time.

The summary that is provided in Table 4 is based on five-run results for the TOP instances. The reason that five runs were performed is the fact that the maximum CPU times of one-run results are significantly lower than the maximum CPU times as reported in Tang and Miller-Hooks (2005). This implies that despite an increase in computation time, performing five repetitions will still not require excessive computation time. Compared with one-run results, of which a summary can be found in Table 7 in Appendix B, performing five runs leads to further improvement of the results for several problem categories. Tables 8-13 contain five-run results of every separate problem instance. Note that in these tables, maximum CPU times are reported for 'Tabu', since five repetitions were performed, whereas CPU times reported under 'AFHS' are based on one-run results.

Table 4: *Summary of five-run results TOP*
This table shows (1) the average reward for all 18 problem categories of the TOP and (2) the maximum CPU times in seconds of all the problems in each category, and (3) the relative difference between the results found and the results found in AFHS.

Set	n	m	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU_{max}	
1	32	4	138	0.6	138	1.5	0.0
3	33	4	347	0.7	353	0.8	-1.7
4	100	4	773	11.8	785	136.8	-1.5
5	66	4	694	4.0	699	22.6	-0.9
6	64	4	712	3.0	713	19.9	-0.1
7	102	4	505	11.9	515	101.0	-2.1
1	32	3	165	0.7	166	2.6	-0.6
3	33	3	622	1.2	634	3.3	-1.9
4	100	3	821	15.4	844	317.4	-3.0
5	66	3	763	5.0	776	51.7	-1.5
6	64	3	785	4.4	787	37.2	-0.3
7	102	3	578	14.4	593	143.2	-2.7
1	32	2	130	1.0	135	1.3	-3.7
3	33	2	433	1.3	441	6.6	-1.8
4	100	2	876	29.2	895	796.7	-2.1
5	66	2	864	8.4	887	71.3	-2.6
6	64	2	811	7.7	818	53.8	-0.9
7	102	2	609	22.2	634	432.6	-3.9

4.2 CTOP results

In Table 5, results of performing one run of the AMP on the benchmark instances of the CTOP are reported. The reason that only one run was performed is that, except for data set 9, the upper bound on possible collected reward was already reached by the algorithm after executing the AMP once. That is, the sum of the rewards of all customers was collected by the vehicles in the CTOP. This is not a surprising result since the maximum number of vehicles m is relatively high, which facilitates visiting every customer by one of these vehicles. As a consequence, the CTOPs solved in data sets 3-16 show strong similarities with the multiple knapsack problem, where n vertices (items) need to be divided over m vehicles (knapsacks). In order to do a better evaluation of the quality of our tabu search procedure, we therefore tested our algorithm on modified test instances; we solved each instance with a maximum of 2, 3 and 4 vehicles. Results of these tests can be found in Table 14 in Appendix B. Summary results are provided in Table 6. We observe that for problem categories with relatively lower values for maximum capacity Q and time limit L, our tabu search procedure performs well compared to the tabu procedure by Archetti et al. (2009); the relative difference between average rewards is always smaller than 4%. However, the tabu heuristic as proposed by Archetti et al. (2009) seems to perform better than ours for higher values of Q and L. Nonetheless, the results in Table 5 show that our AMP and tabu search heuristic performs excellently when a large number of vehicles are available, regardless of the fact that values for Q and L are relatively high. Also, it is important to take note of the substantial differences in maximum computation time of the Tabu algorithm and the AFHS algorithm. From Table 6 we can conclude that the maximum computation time of the AMP with incorporated tabu procedure as described in this paper is at least 35 times lower than that of the AFHS tabu procedure. In addition, the maximum CPU times as reported in the 'Tabu' column encompass the entire AMP, which includes the tabu search procedure, whereas the maximum CPU times as reported under the 'AFHS' column only provides the required time to execute the tabu procedure itself. This difference in measuring computation time also provides an explanation for the observed difference in CPU times between the two algorithms as reported in Table 5. Archetti et al. (2009) report only the computation time of the tabu procedure, which is terminated as soon as the upper bound is reached. On the other hand, the 'Tabu' results show CPU times of the entire AMP, which is always executed as a whole and never terminated intermediately. Please note that direct comparison between the running times remains complicated, since lower CPU times might also be attributed to a stronger computational power of the Intel i5-4210U processor with respect to the Intel Pentium 4 CPU 2.80 GHz and 1.048 GB RAM that was used by Archetti et al. (2009).

Table 5: *One-run results CTOP benchmark instances*
This table shows (1) the obtained reward for all 10 benchmark instances for the CTOP and (2) the maximum CPU times in seconds of all problems in each category. A value in italics means that this value is equal to the upper bound for this problem instance.

Set	n	m	Q	L	Tabu		AFHS	
					Reward	CPU_{max}	Reward	CPU
3	101	15	200	200	*1409*	1.7	*1409*	0
6	51	10	160	200	*761*	0.7	*761*	0
7	76	20	140	160	*1327*	1.4	*1327*	0
8	101	15	200	230	*1409*	1.6	*1409*	0
9	151	10	200	200	2039	25.1	2061	163
10	200	20	200	200	*3048*	6.0	*3048*	0
13	121	15	200	720	*1287*	2.6	*1287*	0
14	101	10	200	1040	*1710*	2.7	*1710*	0
15	151	15	200	200	*2159*	4.6	*2159*	0
16	200	15	200	200	*3066*	9.1	2965	270

Table 6: *Summary of five-run results modified CTOP instances*
This table shows (1) the average reward for 9 problem categories of the CTOP, (2) the maximum CPU times in seconds of all the problems in each category, and (3) the relative difference between the results found and the results found in AFHS.

m	Q	L	Tabu		AFHS		%diff.
			Reward	CPU_{max}	Reward	CPU_{max}	
2	50	50	127	7.1	131	312	-3.7
3	50	50	189	5.8	194	303	-2.6
4	50	50	245	5.7	253	324	-3.0
2	75	75	190	27.7	207	1821	-7.9
3	75	75	280	24.0	297	2156	-5.5
4	75	75	368	23.4	387	2435	-5.0
2	100	100	248	56.0	272	2144	-9.0
3	100	100	369	47.8	399	2606	-7.6
4	100	100	480	41.1	520	2124	-7.6

5 Conclusion

In this paper, we developed a heuristic algorithm to solve the capacitated version of the TOP. The proposed algorithm shares strong similarities with the tabu search algorithm as proposed by Tang and Miller-Hooks (2005). The features of a flexible adaptive memory, alternation between neighbourhood stages and the combination of random and greedy procedures for tour improvement can all be found in the algorithm of Tang and Miller-Hooks (2005). Distinction lies in the addition of a variant of the set packing problem to construct a high-reward, feasible solution from the tours in the adaptive memory. Since this problem is solved exactly, the algorithm has become more demanding with regard to computation time. To compensate this, a random vertex insertion procedure that was used by Tang and Miller-Hooks (2005) was left out, which lead to only minor degradation in the solution quality.

Another major distinctive factor is the ability of the developed algorithm to solve not only TOP instances, but also CTOP instances. Computational experiments revealed that the algorithm performs excellently for

benchmark instances provided by Christofides et al. (1979); the optimal solution was found for 9 out of 10 instances. Modifying these benchmark instances allowed us to solve more diverse and more difficult CTOP problems. For these instances, solving to optimality is an unrealistic goal, but our algorithm does still yield satisfactory results. In particular, from the low computation times we can conclude that we developed an efficient algorithm.

Due to the flexibility of the structure of our AMP and tabu search procedure, the algorithm lends itself for further extensions or specifications. For example, the (C)TOP might become more realistic and practically applicable by considering time-dependent customer rewards. Here, some customers assign more value to quick service than other customers, and they are thus prepared to pay a higher price for quicker service. This case could be studied in combination with time-dependent travel times as well. Also, service times can be taken into consideration and customer demands might be stochastic. Regarding the CTOP, one might investigate the possibility of split delivery, as was studied by Archetti et al. (2014). Lastly, a challenge would be not to consider a deterministic set of customers, but to amend the heuristic to one which can solve an on-line (C)TOP, where not all customer information is available in advance.

References

Archetti, C., Bianchessi, N., Speranza, M. G., and Hertz, A. (2014). The split delivery capacitated team orienteering problem. *Networks*, 63(1):16–33.

Archetti, C., Feillet, D., Hertz, A., and Speranza, M. G. (2009). The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, 60(6):831–842.

Archetti, C., Hertz, A., and Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76.

Boussier, S., Feillet, D., and Gendreau, M. (2007). An exact algorithm for team orienteering problems. *4or*, 5(3):211–230.

Butt, S. E. and Cavalier, T. M. (1994). A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research*, 21(1):101–111.

Butt, S. E. and Ryan, D. M. (1999). An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26(4):427–441.

Chao, I. et al. (1993). Algorithms and solutions to multi-level vehicle routing problems.

Chao, I.-M., Golden, B. L., and Wasil, E. A. (1996a). A fast and effective heuristic for the orienteering problem. *European journal of operational research*, 88(3):475–489.

Chao, I.-M., Golden, B. L., and Wasil, E. L. (1996b). The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474.

Christofides, N., Mingozzi, A., and Toth, P. (1979). The vehicle routing problem. *Combinatorial Optimization*, pages 315–338.

CPLEX (2016). IBM ILOG CPLEX Optimizer 12.7.0 (2016). URL https://www.ibm.com/analytics/cplex-optimizer.

El-Hajj, R., Dang, D.-C., and Moukrim, A. (2016). Solving the team orienteering problem with cutting planes. *Computers & Operations Research*, 74:21–30.

Gendreau, M., Laporte, G., and Semet, F. (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2-3):539–545.

Glover, F. (1989). Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206.

Glover, F. (1990). Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32.

Gueguen, C. (1999). *Méthodes de résolution exacte pour les problèmes de tournées de véhicules*. PhD thesis, Châtenay-Malabry, Ecole centrale de Paris.

Ke, L., Archetti, C., and Feng, Z. (2008). Ants can solve the team orienteering problem. *Computers & Industrial Engineering*, 54(3):648–665.

Laporte, G. and Martello, S. (1990). The selective travelling salesman problem. *Discrete applied mathematics*, 26(2-3):193–207.

Olivera, A. and Viera, O. (2007). Adaptive memory programming for the vehicle routing problem with multiple trips. *Computers & Operations Research*, 34(1):28–47.

Taillard, É. D., Gambardella, L. M., Gendreau, M., and Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1–16.

Tang, H. and Miller-Hooks, E. (2005). A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, 32(6):1379–1407.

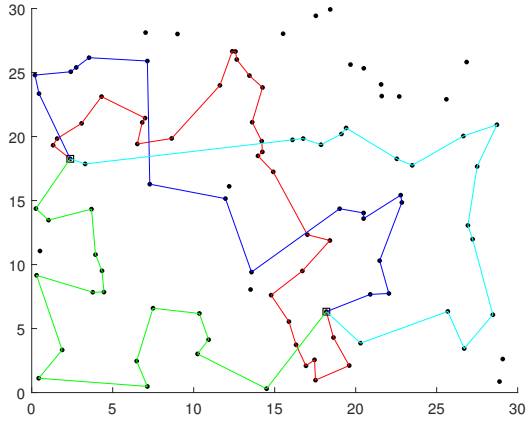
Tarantilis, C. D., Stavropoulou, F., and Repoussis, P. P. (2013). The capacitated team orienteering problem: a bi-level filter-and-fan method. *European Journal of Operational Research*, 224(1):65–78.

Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809.

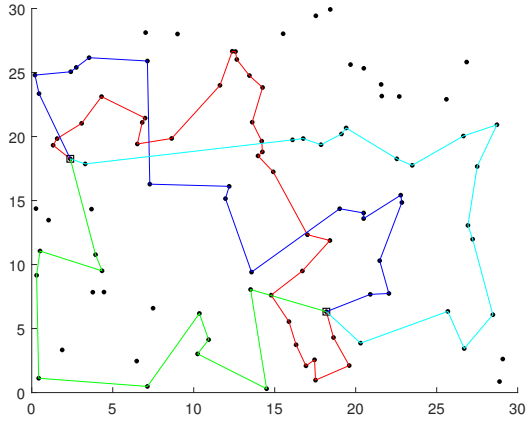
Vansteenwegen, P., Souffriau, W., Berghe, G. V., and Van Oudheusden, D. (2009). Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290.

Appendix

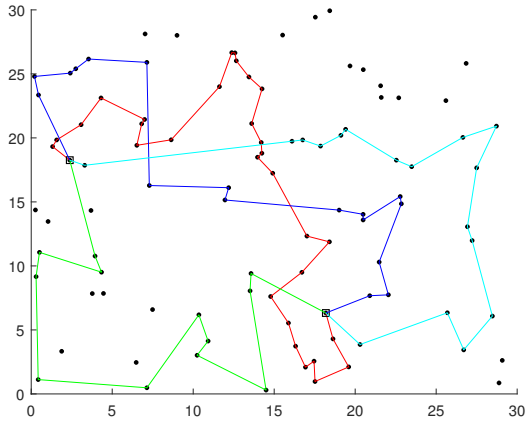
A Illustration of the tabu procedure



(A) Initial solution.

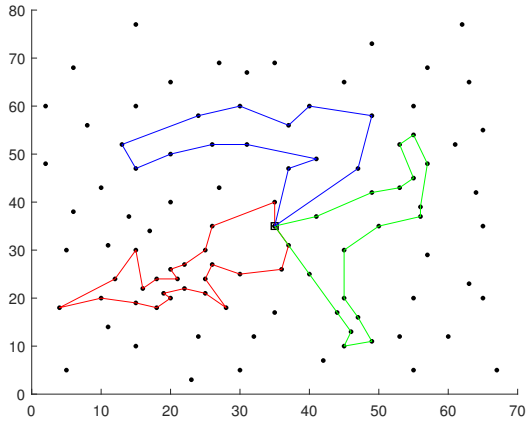


(B) Solution after Step B.3 of the tabu procedure.

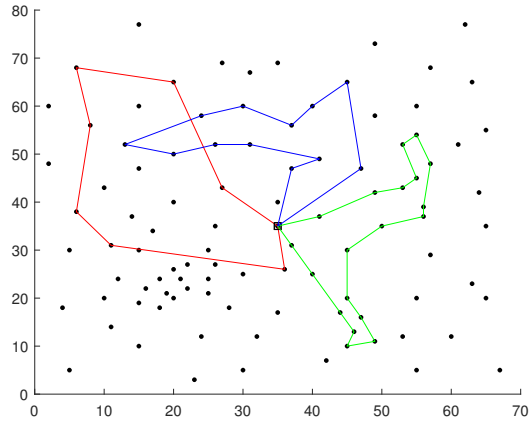


(C) Solution after Step B.4 of the tabu procedure.

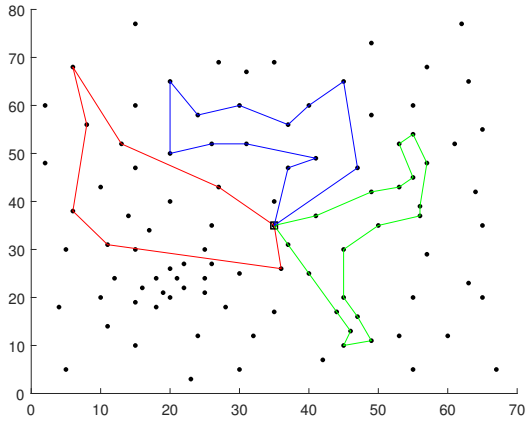
FIGURE 1: An illustration of the routes contained in the solution during the tabu search procedure. In this particular case, Step B.2 and was not successful, so Step B.3 was executed. Step B.4 was successful, so Step B.5 was skipped.



(A) Initial solution.



(B) Solution after Step B.3 of the tabu procedure.



(C) Solution after Step B.5 of the tabu procedure.

FIGURE 2: An illustration of the routes contained in the solution during the tabu search procedure. In this particular case, Step B.2 and Step B.4 were not successful, so Step B.3 and B.5 were executed.

B Results for the TOP

Table 7: *Summary of one-run results TOP*
This table shows (1) the average reward for all 18 problem categories of the TOP and (2) the maximum CPU times in seconds of all the problems in each category.

Set	n	m	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU_{max}	
1	32	4	138	0.6	138	1.5	0.0
3	33	4	347	0.7	353	0.8	-1.7
4	100	4	768	11.8	785	136.8	-2.2
5	66	4	692	4.0	699	22.6	-1.0
6	64	4	709	3.0	713	19.9	-0.6
7	102	4	503	11.9	515	101.0	-2.3
1	32	3	165	0.7	166	2.6	-0.6
3	33	3	618	1.2	634	3.3	-2.5
4	100	3	813	15.4	844	317.4	-3.7
5	66	3	762	5.0	776	51.7	-1.8
6	64	3	784	4.4	787	37.2	-0.4
7	102	3	575	14.4	593	143.2	-3.0
1	32	2	130	1.0	135	1.3	-3.7
3	33	2	431	1.3	441	6.6	-2.3
4	100	2	872	29.2	895	796.7	-2.6
5	66	2	863	8.4	887	71.3	-2.7
6	64	2	811	7.7	818	53.8	-0.9
7	102	2	608	22.2	634	432.6	-4.1

Table 8: *Five-run results TOP data set 1*
This table shows (1) the reward for all problem instances of the TOP in data set 1, (2) the maximum CPU times in seconds of all instances for 'Tabu', and CPU times in seconds of all instances for 'TMH', and (3) the relative difference between our results and the results found by 'TMH'.

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
32	4	18.8	175	0.5	175	1.5	0.0
		18.2	165	0.6	165	1.3	0.0
		12.5	75	0.5	75	0.8	0.0
	3	25.0	215	0.7	220	1.5	-2.3
		24.3	200	0.7	205	2.6	-2.4
		21.7	175	0.7	170	1.4	2.9
		13.3	70	0.5	70	0.8	0.0
	2	23.0	130	1.0	135	1.3	-3.7

Table 9: *Five-run results TOP data set 3*
This table shows (1) the reward for all problem instances of the TOP in data set 3, (2) the maximum CPU times in seconds of all instances for 'Tabu', and CPU times in seconds of all instances for 'TMH', and (3) the relative difference between our results and the results found by 'TMH'.

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
33	4	22.5	540	0.7	560	0.7	-3.6
		15.0	310	0.6	310	0.8	0.0
		10.0	190	0.7	190	0.6	0.0
	3	36.7	730	1.2	750	3.3	-2.7
		31.7	650	0.7	680	3.1	-4.4
		30.0	640	0.8	640	2.1	0.0
		28.3	580	0.7	590	2.0	-1.7
		25.0	510	0.8	510	2.0	0.0
	2	47.5	730	1.3	760	5.4	-3.9
		42.5	680	1.3	690	6.6	-1.4
		30.0	500	1.2	490	1.5	2.0
		27.5	440	1.2	460	3.8	-4.3
		25.0	390	1.7	410	3.1	-4.9
		20.0	290	0.9	290	1.2	0.0
		17.5	250	0.9	250	0.8	0.0
		12.5	180	0.9	180	1.2	0.0

Table 10: *Five-run results TOP data set 4*
This table shows (1) the reward for all problem instances of the TOP in data set 4, (2) the maximum CPU times in seconds of all instances for 'Tabu', and CPU times in seconds of all instances for 'TMH', and (3) the relative difference between our results and the results found by 'TMH'.

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
100	4	60.0	1233	9.7	1255	136.8	-1.8
		57.5	1203	10.7	1243	86.6	-3.2
		55.0	1135	9.5	1165	79.7	-2.6
		52.5	1096	10.4	1124	115.5	-2.5
		50.0	1045	10.1	1056	134.5	-1.0
		47.5	994	10.0	1014	112.8	-2.0
		45.0	934	12.7	977	111.9	-4.4
		42.5	893	10.5	910	78.5	-1.9
		40.0	860	10.3	875	96.2	-1.7
		37.5	808	9.9	819	80.9	-1.3
		35.0	730	8.9	732	63.8	-0.3
		32.5	648	9.0	627	47.3	3.3
		30.0	558	7.9	554	31.4	0.7
		27.5	461	8.3	453	23.7	1.8
		25.0	324	7.8	315	11.2	2.9
		22.5	183	6.9	182	3.2	0.5
		20.0	38	9.4	38	1.4	0.0

Table 10: *Five-run results TOP data set 4* (continued)

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
	3	80.0	1282	14.3	1288	241.1	-0.5
		76.7	1246	14.6	1282	317.4	-2.8
		73.3	1228	13.3	1265	220.9	-2.9
		70.0	1192	15.5	1249	210.9	-4.6
		66.7	1149	15.7	1218	296.2	-5.7
		63.3	1102	14.2	1151	193.7	-4.3
		60.0	1052	14.5	1119	143.9	-6.0
		56.7	990	15.9	1005	167.3	-1.5
		53.3	931	13.7	951	137.0	-2.1
		50.0	872	13.8	906	164.9	-3.8
		46.7	824	12.6	860	169.4	-4.2
		43.3	768	15.8	785	92.3	-2.2
		40.0	703	12.2	709	134.1	-0.8
		36.7	629	10.7	646	50.8	-2.6
		33.3	565	11.3	579	43.2	-2.4
		30.0	458	9.9	465	56.5	-1.5
		26.7	335	9.2	333	22.3	0.6
		23.3	193	7.7	192	15.3	0.5
		20.0	38	9.4	38	1.4	0.0
	2	120.0	1282	25.8	1306	457.1	-1.8
		115.0	1270	27.6	1294	796.7	-1.9
		110.0	1241	24.3	1277	579.3	-2.8
		105.0	1212	26.8	1255	766.9	-3.4
		100.0	1172	25.8	1208	650.3	-3.0
		95.0	1134	21.7	1175	384.2	-3.5
		90.0	1103	24.3	1150	317.1	-4.1
		85.0	1058	22.4	1089	464.2	-2.8
		80.0	1019	23.7	1022	317.1	-0.3
		75.0	971	19.7	963	342.4	0.8
		70.0	917	23.2	914	232.6	0.3
		65.0	855	21.7	915	332.8	-6.6
		60.0	803	20.7	827	173.7	-2.9
		55.0	726	19.7	749	114.0	-3.1
		50.0	667	16.8	666	163.3	0.2
		45.0	592	17.2	593	92.3	-0.2
		40.0	515	14.8	517	72.1	-0.4
		35.0	441	13.6	438	58.9	0.7
		30.0	331	12.3	341	90.2	-2.9
		25.0	206	9.5	202	33.7	2.0

Table 11: *Five-run results TOP data set 5*
This table shows (1) the reward for all problem instances of the TOP in data set 5, (2) the maximum CPU times in seconds of all instances for 'Tabu', and CPU times in seconds of all instances for 'TMH', and (3) the relative difference between our results and the results found by 'TMH'.

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
66	4	32.5	1620	3.3	1575	12.5	2.9
		31.2	1520	3.8	1520	22.6	0.0
		30.0	1380	3.3	1410	13.3	-2.1
		28.8	1360	3.1	1380	14.1	-1.4
		27.5	1260	3.7	1310	19.1	-3.8
		26.2	1200	2.9	1275	13.1	-5.9
		25.0	1100	3.9	1100	7.9	0.0
		23.8	1000	3.6	1000	14.6	0.0
		22.5	925	4.0	960	8.5	-3.6
		21.2	860	3.8	860	7.4	0.0
		20.0	760	4.0	760	6.8	0.0
		18.8	680	4.3	680	5.9	0.0
		17.5	620	3.7	620	5.7	0.0
		16.2	555	3.3	555	5.9	0.0
		15.0	430	3.0	430	4.9	0.0
		13.8	340	3.1	340	5.0	0.0
		12.5	340	2.9	340	2.9	0.0
		11.2	240	3.2	240	2.3	0.0
		10.0	140	2.7	140	1.8	0.0
		8.8	140	3.5	140	1.6	0.0
		7.5	80	2.8	80	1.4	0.0
		6.2	50	2.6	20	0.1	150.0
		5.0	20	2.2	20	0.1	0.0
		3.8	20	2.5	20	0.3	0.0
	3	43.3	1635	4.7	1635	51.7	0.0
		41.7	1590	4.6	1580	29.2	0.6
		40.0	1505	4.7	1530	26.0	-1.6
		38.3	1430	4.5	1465	26.8	-2.4
		36.7	1400	4.5	1410	26.0	-0.7
		35.0	1290	4.3	1330	39.1	-3.0
		33.3	1220	5.0	1240	19.9	-1.6
		31.7	1145	5.3	1175	19.5	-2.6
		30.0	1055	4.7	1115	25.0	-5.4
		28.3	1025	4.3	1065	17.2	-3.8
		26.7	930	4.2	990	11.5	-6.1
		25.0	830	4.4	835	10.7	-0.6
		23.3	755	4.9	755	9.6	0.0
		21.7	650	5.1	650	8.3	0.0
		20.0	585	5.0	575	7.8	1.7

Table 11: *Five-run results TOP data set 5* (continued)

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
		18.3	495	4.3	495	7.5	0.0
		16.7	470	4.0	470	5.7	0.0
		15.0	335	3.4	335	6.3	0.0
		13.3	260	3.2	260	6.0	0.0
		11.7	185	3.5	185	8.0	0.0
		10.0	110	3.3	110	5.5	0.0
		8.3	95	3.4	95	1.7	0.0
		6.7	60	3.5	60	1.9	0.0
		5.0	20	2.8	20	0.1	0.0
		3.3	15	2.9	15	1.1	0.0
	2	65.0	1665	8.3	1665	63.2	0.0
		62.5	1640	8.7	1630	53.9	0.6
		60.0	1600	8.7	1610	71.3	-0.6
		57.5	1560	9.2	1560	52.0	0.0
		55.0	1480	9.9	1500	31.4	-1.3
		52.5	1440	8.5	1445	36.5	-0.3
		50.0	1320	8.1	1380	34.9	-4.3
		47.5	1270	8.3	1310	33.9	-3.1
		45.0	1200	7.0	1260	46.4	-4.8
		42.5	1100	6.9	1185	51.4	-7.2
		40.0	1020	7.9	1090	53.6	-6.4
		37.5	940	6.8	975	30.1	-3.6
		35.0	880	6.6	920	21.6	-4.3
		32.5	820	6.9	860	23.8	-4.7
		30.0	710	6.9	770	18.3	-7.8
		27.5	640	6.7	670	13.8	-4.5
		25.0	560	6.2	560	11.7	0.0
		22.5	470	6.1	480	12.2	-2.1
		20.0	400	5.3	410	10.8	-2.4
		17.5	320	5.9	320	7.6	0.0
		15.0	240	4.2	240	5.3	0.0
		12.5	180	4.5	180	3.9	0.0
		10.0	80	3.4	80	2.8	0.0
		7.5	50	3.1	50	1.6	0.0
		5.0	20	3.2	20	1.0	0.0

Table 12: *Five-run results TOP data set 6*
This table shows (1) the reward for all problem instances of the TOP in data set 6, (2) the maximum CPU times in seconds of all instances for 'Tabu', and CPU times in seconds of all instances for 'TMH', and (3) the relative difference between our results and the results found by 'TMH'.

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
64	4	20.0	1062	3.1	1068	8.9	-0.6
		18.8	912	3.0	912	11.0	0.0
		17.5	690	3.1	696	8.7	-0.9
		16.2	528	3.8	522	4.4	1.1
		15.0	366	3.4	366	3.2	0.0
	3	26.7	1152	4.0	1152	18.6	0.0
		25.0	1062	4.1	1080	15.4	-1.7
		23.3	990	4.2	990	14.5	0.0
		21.7	888	4.7	876	10.8	1.4
		20.0	828	4.8	828	10.1	0.0
		18.3	636	3.6	612	15.0	3.9
		16.7	444	3.6	444	7.5	0.0
		15.0	282	3.9	282	3.5	0.0
	2	40.0	1224	7.6	1260	45.7	-2.9
		37.5	1176	7.2	1188	30.3	-1.0
		35.0	1092	7.1	1116	26.3	-2.2
		32.5	1026	6.8	1032	20.0	-0.6
		30.0	936	7.1	936	17.6	0.0
		27.5	888	6.2	888	16.7	0.0
		25.0	780	5.6	780	14.1	0.0
		22.5	660	5.0	660	12.1	0.0
		20.0	588	5.8	588	11.1	0.0
		17.5	360	5.2	360	8.3	0.0
		15.0	192	4.0	192	4.5	0.0

Table 13: *Five-run results TOP data set 7*
This table shows (1) the reward for all problem instances of the TOP in data set 7, (2) the maximum CPU times in seconds of all instances for 'Tabu', and CPU times in seconds of all instances for 'TMH', and (3) the relative difference between our results and the results found by 'TMH'.

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
102	4	100.0	1027	10.5	1067	86.6	-3.7
		95.0	976	12.5	1019	84.3	-4.2
		90.0	923	11.9	966	101.0	-4.5
		85.0	890	11.4	905	95.2	-1.7
		80.0	818	11.2	832	82.0	-1.7
		75.0	770	11.2	776	71.3	-0.8
		70.0	701	11.3	726	54.4	-3.4
		65.0	643	9.9	643	68.6	0.0
		60.0	571	9.6	576	31.8	-0.9

Table 13: *Five-run results TOP data set 7* (continued)

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
		55.0	507	8.4	503	44.9	0.8
		50.0	456	7.4	462	23.6	-1.3
		45.0	359	7.5	359	20.6	0.0
		40.0	285	7.5	285	17.2	0.0
		35.0	217	5.5	217	10.0	0.0
		30.0	164	7.0	164	5.5	0.0
		25.0	123	6.9	123	3.3	0.0
		20.0	79	6.0	79	0.1	0.0
		15.0	46	5.7	46	0.1	0.0
		10.0	30	6.0	30	0.1	0.0
	3	133.3	1062	15.5	1098	143.2	-3.3
		126.7	1018	14.2	1061	99.8	-4.1
		120.0	961	14.7	1011	93.6	-4.9
		113.3	940	13.7	966	98.8	-2.7
		106.7	893	13.7	922	74.0	-3.1
		100.0	841	15.2	874	102.8	-3.8
		93.3	780	14.7	789	126.5	-1.1
		86.7	732	14.2	756	121.2	-3.2
		80.0	672	13.5	681	69.5	-1.3
		73.3	614	12.9	632	94.4	-2.8
		66.7	547	12.1	563	107.7	-2.8
		60.0	479	10.7	481	36.0	-0.4
		53.3	422	9.7	416	34.0	1.4
		46.7	344	7.9	344	21.1	0.0
		40.0	248	7.9	247	23.7	0.4
		33.3	175	6.9	175	12.0	0.0
		26.7	117	6.4	117	3.7	0.0
		20.0	79	7.5	79	2.3	0.0
		13.3	46	5.6	46	0.1	0.0
	2	200.0	1079	26.2	1165	290.6	-7.4
		190.0	1043	25.9	1116	215.6	-6.5
		180.0	1011	22.0	1067	432.6	-5.2
		170.0	987	24.7	1017	239.6	-2.9
		160.0	941	23.6	987	272.1	-4.7
		150.0	901	21.5	914	202.8	-1.4
		140.0	829	21.2	864	224.3	-4.1
		130.0	787	25.6	817	174.1	-3.7
		120.0	729	19.4	767	217.5	-5.0
		110.0	670	19.0	702	120.1	-4.6
		100.0	611	20.5	638	118.7	-4.2
		90.0	566	18.9	578	84.4	-2.1
		80.0	514	17.6	521	52.0	-1.3

Table 13: *Five-run results TOP data set 7* (continued)

n	m	L	Tabu		TMH		%diff.
			Reward	CPU_{max}	Reward	CPU	
		70.0	457	14.7	459	74.6	-0.4
		60.0	381	15.2	382	42.7	-0.3
		50.0	290	9.6	290	37.7	0.0
		40.0	190	10.3	190	16.3	0.0
		30.0	101	7.7	101	8.6	0.0
		20.0	64	7.4	64	2.8	0.0
		10.0	30	4.3	30	0.1	0.0

C Results for the CTOP

Table 14: *Five-run results modified CTOP instances*
This table shows (1) the reward for all modified CTOP instances, (2) the maximum CPU times in seconds of all instances for 'Tabu', and CPU times in seconds of all instances for 'AFHS', and (3) the relative difference between the results found and the results found by 'AFHS'.

Instance	n	m	Q	L	Tabu		AFHS		%diff.
					Reward	CPU_{max}	Reward	CPU	
3	101	2	50	50	128	0.8	133	34	-3.8
		3	50	50	193	0.7	198	34	-2.5
		4	50	50	253	0.7	260	35	-2.7
		2	75	75	193	3.4	208	224	-7.2
		3	75	75	287	2.8	307	225	-6.5
		4	75	75	380	2.2	403	299	-5.7
		2	100	100	252	5.7	277	291	-9.0
		3	100	100	375	5.0	408	320	-8.1
		4	100	100	489	3.9	531	317	-7.9
6	51	2	50	50	118	0.3	121	3	-2.5
		3	50	50	177	0.3	177	3	0.0
		4	50	50	218	0.2	222	3	-1.8
		2	75	75	179	0.5	183	33	-2.2
		3	75	75	263	0.5	269	28	-2.2
		4	75	75	338	0.5	348	25	-2.9
		2	100	100	234	1.0	252	28	-7.1
		3	100	100	347	0.8	369	30	-6.0
		4	100	100	452	0.7	482	25	-6.2
7	76	2	50	50	126	0.4	126	14	0.0
		3	50	50	187	0.5	187	14	0.0
		4	50	50	240	0.4	240	13	0.0
		2	75	75	183	1.1	193	89	-5.2

Table 14: *Five-run results modified CTOP instances* (continued)

Instance	n	m	Q	L	Tabu		AFHS		%diff.
					Reward	CPU_{max}	Reward	CPU	
		3	75	75	270	1.1	287	87	-5.9
		4	75	75	358	0.9	378	88	-5.3
		2	100	100	238	2.1	266	95	-10.5
		3	100	100	354	1.8	397	113	-10.8
		4	100	100	472	1.6	521	119	-9.4
8	101	2	50	50	128	0.8	133	34	-3.8
		3	50	50	192	0.7	198	34	-3.0
		4	50	50	253	0.7	260	35	-2.7
		2	75	75	197	3.3	208	224	-5.3
		3	75	75	289	2.8	307	225	-5.9
		4	75	75	378	2.3	403	299	-6.2
		2	100	100	253	5.7	277	2291	-8.7
		3	100	100	374	4.5	409	320	-8.6
		4	100	100	495	4.0	531	317	-6.8
9	151	2	50	50	131	2.4	137	115	-4.4
		3	50	50	193	2.2	201	115	-4.0
		4	50	50	252	1.9	262	112	-3.8
		2	75	75	196	11.0	210	785	-6.7
		3	75	75	293	7.2	310	808	-5.5
		4	75	75	386	8.6	407	958	-5.2
		2	100	100	258	19.7	278	971	-7.2
		3	100	100	378	17.0	414	1521	-8.7
		4	100	100	494	15.2	539	924	-8.3
10	200	2	50	50	128	7.1	134	312	-4.5
		3	50	50	192	5.8	200	303	-4.0
		4	50	50	254	5.7	265	324	-4.2
		2	75	75	200	20.6	208	1759	-3.8
		3	75	75	295	15.7	310	1890	-4.8
		4	75	75	394	18.3	410	2194	-3.9
		2	100	100	259	42.9	280	2115	-7.5
		3	100	100	393	37.1	417	2606	-5.8
		4	100	100	507	36.0	549	2077	-7.7
13	121	2	50	50	119	1.5	134	10	-11.2
		3	50	50	180	1.1	193	9	-6.7
		4	50	50	228	0.9	243	8	-6.2
		2	75	75	177	1.8	193	4	-8.3
		3	75	75	250	1.2	265	4	-5.7
		4	75	75	316	0.9	323	4	-2.2
		2	100	100	221	3.7	253	27	-12.6
		3	100	100	326	2.6	344	24	-5.2

Table 14: *Five-run results modified CTOP instances* (continued)

Instance	n	m	Q	L	Tabu		AFHS		%diff.
					Reward	CPU_{max}	Reward	CPU	
		4	100	100	407	1.7	419	24	-2.9
14	101	2	50	50	124	0.9	124	37	0.0
		3	50	50	184	0.9	184	35	0.0
		4	50	50	235	0.7	241	36	-2.5
		2	75	75	173	2.9	190	98	-8.9
		3	75	75	257	2.2	279	97	-7.9
		4	75	75	337	2.1	366	102	-7.9
		2	100	100	250	4.9	271	181	-7.7
		3	100	100	370	4.9	399	248	-7.3
		4	100	100	486	4.8	523	210	-7.1
15	151	2	50	50	130	2.4	134	123	-3.0
		3	50	50	196	2.2	200	129	-2.0
		4	50	50	257	2.1	266	113	-3.4
		2	75	75	202	11.2	211	904	-4.3
		3	75	75	300	9.1	315	782	-4.8
		4	75	75	397	8.0	414	998	-4.1
		2	100	100	261	17.9	282	924	-7.4
		3	100	100	387	18.0	416	883	-7.0
		4	100	100	497	14.7	549	1252	-9.5
16	200	2	50	50	133	5.3	137	305	-2.9
		3	50	50	197	4.7	203	303	-3.0
		4	50	50	261	4.5	269	295	-3.0
		2	75	75	203	27.7	212	1821	-4.2
		3	75	75	300	24.0	317	2156	-5.4
		4	75	75	396	23.4	420	2435	-5.7
		2	100	100	250	56.0	285	2144	-12.3
		3	100	100	384	47.8	421	2421	-8.8
		4	100	100	502	41.1	554	2124	-9.4