Erasmus School of Economics

Implementation of an iterated local search heuristic for the team orienteering problem with time windows

Author:
Renée Buijs
345246

Supervisor:
Dr. Dennis Huisman

Bachelor thesis
Econometrics and Operations Research
Erasmus University Rotterdam
July 3, 2015

Abstract

When tourists visit a city or region, they cannot visit every possible attraction as they are constrained in time. A Personalized Electronic Tourist guide may be used to derive a personalized tourist route that maximizes the tourists' satisfaction. The planning problem that needs to be solved can be modeled as a Team Orienteering Problem with Time Windows (TOPTW). In the TOPTW, a set of locations is given, each with a score, a service time and a time window. The goal is to maximize the sum of the collected scores by a fixed number of routes, while the visits are within the time windows of the locations and the time budget of the tourist. Each route can be interpreted as a day trip. In this thesis, we discuss the algorithm used to solve the TOPTW developed by Vansteenwegen et al. (2009). They developed an iterated local search heuristic with an insertion step and a shake step. We implement this heuristic and we compare the results. Moreover, we extend the problem by adding the possibility to add more constraints, such as a limited money budget. This problem can be solved as a Multi Constrained Team Orienteering Problem with Time Windows (MCTOPTW). To solve a MCTOPTW, we make the adjustments to the heuristic proposed by Garcia et al. (2010) and we compare the results.

Table of Contents

1 Introduction

Most tourists visiting a city want to visit as many attractions as possible. If a tourist wants to visit every tourist attraction during a city trip in a large city, a considerable amount of time is required. However, most tourists visit a city during one or more days which means the amount of time is limited. Therefore, the tourist has to make a selection of what he believes to be the most valuable attractions. Once the selection is made, the tourist decides on a route, keeping in mind the available time and the opening hours of the attractions. This part, of making a feasible route in order to visit the most attractions within the available time, is difficult. Furthermore, when the tourist deviates from the original route and the original route becomes infeasible, he has to start from scratch to find an optimal route for the remaining part of the trip.

A Personalized Electronic Tourist guide (PET) may be used to derive a personalized tourist route. This is a mobile hand-held device that creates a route that maximizes the tourists' satisfaction, taking into account several restrictions, such as the opening and closing hours of the attractions, the service time, the available time and the travel distances between the attractions. The PET has to solve Tourist Trip Design Problems (TTDP). In this paper, we consider the Team Orienteering Problem with Time Windows (TOPTW). The TOPTW is a simplified version of the TTDP, which takes the limited time budget of the tourist into account. In the TOPTW a set of locations is given, each with a score, a service time and a time window. The goal is to maximize the sum of the collected scores by a fixed number of routes. Each route can be interpreted as a day trip.

When a tourist faces an unexpected event or wants to change the plans, he needs a new route. He does not want to wait too long to receive a new route in such situations. For instance, when a tourist did not like a attraction and is done earlier, he does not want to wait minutes for a modified plan to become available. This means the computation time must be limited.

In this thesis, we discuss the algorithm developed by Vansteenwegen et al. (2009). This algorithm, that obtains high quality results in limited amount of time, makes use of an insertion and a shake step. We reproduce the results of the article. Moreover, we discuss the adjustments to the model proposed by Garcia et al. (2010) to solve the Multi Constrained Team Orienteering Problem with Time Windows (MCTOPTW). In this model, it is possible to add more constraints to the problem.

The thesis is structured as follows. In the next chapter we present a literature review. In chapter 3, a mathematical problem is presented. In chapter 4, the heuristic and the two steps of the heuristic are described. We illustrate the steps with a small example. The results are discussed in chapter 5. In chapter 6, we discuss the adjustments to the heuristic to solve the MCTOPTW and we show the results. The conclusions and topics of future research are discussed in chapter 7.

2 Literature review

In this chapter, we briefly discuss relevant literature about the team orienteering problem with time windows.

Many articles have been published regarding the (team) orienteering problem without time windows. Kantor and Rosenwein (1992) were the first to solve the orienteering problem with time windows. They model it as a problem on a graph, with a set of nodes (customers), each with an associated profit and service duration (time window), and a set of arcs, each with an associated travel time. They construct an acyclic path beginning at a specified origin and ending at a specified destination that maximizes the total profit while taking into account the time windows and time budget. They use a so called tree heuristic, that systematically generates a list of feasible paths and then selects the most profitable path from the list. The tree heuristic produces improved values of total profit for heavily-constrained, modest-sized problems with computational effort in comparison with an insertion heuristic.

Labadi et al. (2012) developed a local search heuristic algorithm for TOPTW based on a variable neighborhood structure. They propose a variable neighborhood search procedure in which a segment of a path is replaced by nodes offering more profit. Based on the solution of the assignment problem related to the TOPTW, the algorithm decides which arcs to select.

Lin et al. (2012) presents a simulated annealing based heuristic approach for the TOPTW. In each iteration, a neighboring solution is obtained from the current solution by applying a swap, insertion or inversion swap. A new solution is adopted if it is more profitable than the current one. If the solution is not more profitable, the new solution might again replace the current one with a probability inversely proportional to the difference in profits between the old and new solution. After applying this procedure for a specific number of iterations, local search is applied to improve the best solution so far.

Montemanni and Gambardella (2009) discusses an Ant Colony System (ACS) based method. This method takes advantage of a solution model based on a hierarchic generalization of the original problem, which is combined with the ACS algorithm. The quality of solutions is high, but at the expense of long computation times.

Vansteenwegen et al. (2009) proposed an iterated local search heuristic

(ILS). This is the fastest known algorithm proposed for TOPTW (Vansteenwegen et al., 2011). This algorithm is discussed in this thesis.

Garcia et al. (2010) adjusted the heuristic of Vansteenwegen et al. (2009) to solve the multi constrained team orienteering problem with time windows (MCTOPTW). Also this heuristic is discussed in this thesis.

3 Mathematical program

In this chapter, we discuss the formulation of the team orienteering problem with time windows as a mathematical program.

The team orienteering problem with time windows can be described as follows. A set of n locations is given, $i = 1, ..., n$. The route starts and ends at the same location. Every location i is assigned a score S_i, a service or visiting time T_i and a time window for the starting time of the service $[O_i, C_i]$. For all locations, the time c_{ij} needed to travel from location i to location j is known. The time is limited to a given time budget T_{max}. The objective of the OPTW is to determine a single route which maximizes the total collected score, where some of the locations are visited during the time windows within the time budget T_{max}. Each location can be visited at most once. It is allowed to wait at a location before its time windows starts. The TOPTW is an OPTW where the goal is to determine m routes, each limited by T_{max}, that maximizes the total collected score.

The TOPTW can be formulated as an integer program. We define the following decision variables:

- $x_{ijd} = 1$ if, in route d, a visit to location i is followed by a visit to location j, 0 otherwise.
- $y_{id} = 1$ if location i is visited in route d, 0 otherwise.
- s_{id} is the start of the service at location i in route d.

Moreover, M is a large constant.

As formulated by Vansteenwegen et al., the TOPTW formulation is as follows:

$$Max \sum_{d=1}^{m} \sum_{i=2}^{n-1} S_i y_{id} \tag{1}$$

$$\sum_{d=1}^{m} \sum_{j=2}^{n-1} x_{1jd} = \sum_{d=1}^{m} \sum_{i=2}^{n-1} x_{ind} = m \tag{2}$$

$$\sum_{i=1}^{n-1} x_{ikd} = \sum_{j=2}^{n} x_{jkd} = y_{kd} \quad (k = 2, ..., n-1; d = 1, ..., m) \tag{3}$$

$$s_{id} + T_i + c_{ij} - s_{jd} \leq M(1 - x_{ijd}) \quad (i, j = 1, ..., n; d = 1, ..., m) \tag{4}$$

$$\sum_{d=1}^{m} y_{kd} \leq 1 \quad (k = 2, ..., n - 1) \tag{5}$$

$$\sum_{i=1}^{n-1} (T_i y_{id} + \sum_{j=2}^{n} c_{ij} x_{ijd}) \leq T_{max} \quad (d = 1, ..., m) \tag{6}$$

$$O_i \leq s_{id} \quad (i = 1, ..., n; d = 1, ...m) \tag{7}$$

$$s_{id} \leq C_i \quad (i = 1, ..., n; d = 1, ...m) \tag{8}$$

$$x_{ijd}, y_{id} \in \{0, 1\} \quad (i, j = 1, ..., n; d = 1, ..., m) \tag{9}$$

The objective function (1) maximizes the total collected score. Constraints (2) guarantee that all tours start from location 1 and end at location n. Constraints (3) ensure that all locations, except the starting point and ending point, have one location before and one location after visited. Constraints (4) determine the timeline of each tour. Constraints (5) ensure that every location is visited at most once. Constraints (6) limit the time budget. Constraints (7) ensure that the service at a location can only start after the start of the time window. Constraints (8) ensure that the service at a location can only start before the end of the time window.

4 Iterated local search heuristic

The TOPTW is a highly constrained problem and very difficult to solve. For the personalized electronic tourist guide, it is required to solve TOPTW with high quality in a few seconds. The OP is NP-hard, which means that it is unlikely that the TOPTW can be solved to optimality within polynomial time. Therefore, a fast iterated local search heuristic has been developed. This heuristic combines an insertion step and a shake step to escape from local optima. In this chapter, we first discuss the insertion step and shake step. Both steps are illustrated by a small example. After this, we discuss the iterated local search heuristic.

4.1 Example

To illustrate the insertion step and shake step, we make use of a small example. This example takes as input four locations, each assigned a service time, a score and a time window, as can be seen in Table 1. Location 0 is the start and ending point of the tour. T_{max} is equal to the end of time window of location 0, which is 30 in this case. The time to travel between the locations is shown in Table 2.

Table 1: Data locations of example

Location	Service time	Score	Start time window	End time window
0	0	0	0	30
1	7	14	10	25
2	4	4	4	40
3	6	8	5	20

Table 2: Travel times between locations

Location	0	1	2	3
0	0	3	4	5
1	3	0	5	4
2	4	5	0	3
3	5	4	3	0

4.2 Insertion step

In each insertion step, a visit is inserted in the tour. A visit can only be inserted in a tour if all visits scheduled after the insertion place still satisfy their time window and if the time budget of the tour is not violated. By recording the two additional variables $Wait$ and $MaxDelay$ for each included location, the visits can be checked on their feasibility. $Wait$ is defined as the waiting time in case the arrival at a location takes place before the start of the time window of that location. The service can start when the time window opens. If the arrival takes place during the time window, there is no need to wait, which means $Wait$ equals zero.

$$Wait_i = \max[0, O_i - Arrival_i] \tag{10}$$

$MaxDelay$ is defined as the maximum time the service completion of a given visit can be delayed, without making any visit in the tour infeasible. If $MaxDelay$ of location i is limited by its own time window, it is equal to the end of the time window minus the service time. If $MaxDelay$ of location i is not limited by its own time window, it is equal to the sum of $Wait$ and $MaxDelay$ of the next location $i + 1$:

$$MaxDelay_i = \min[C_i - s_i, Wait_{i+1} + MaxDelay_{i+1}] \tag{11}$$

$TimeInsertion$ is defined as the total time consumption to insert an extra visit j between visits i and k:

$$TimeInsertion_j = c_{ij} + Wait_j + T_j + c_{jk} - c_{ik} \tag{12}$$

An insertion is only feasible if the total time consumption does not exceed the sum of $Wait_k$ and $MaxDelay_k$ of visit k. This gives the following formula to check feasibility:

$$TimeInsertion_j = c_{ij} + Wait_j + T_j + c_{jk} - c_{ik} \leq Wait_k + MaxDelay_k$$

The arrival time ($Arrival$) and start of the service ($Start$) are recorded. An insertion is only feasible if the start of the service of location j is within the time window of location j:

$$O_j \leq Start_j \leq C_j \tag{13}$$

For each visit the best possible insert position, with the lowest $TimeInsertion$ is determined. For each visit a ratio is calculated:

$$Ratio_i = \frac{S_i^2}{TimeInsertion_i} \tag{14}$$

The ratio represents a measure how profitable it is to visit location i versus the time delay this visit incurs. The visit with the highest ratio will be selected for insertion. The score is applied to a power two because, due to waiting and time windows, insertion time becomes less relevant than the score for adding new visits to the tour.

Algorithm 1 presents the pseudo code for the insertion step. Firstly, for each non included visit $TimeInsertion$ is calculated for every position in the tour. We take the position with the smallest $TimeInsertion$ and check if it is feasible to add the visit at this position. If it is feasible, we calculate the *Ratio*. We take the visit with the highest *Ratio* and we insert this visit in the tour. The *Arrival*, *Start* and $Wait$ for the inserted visit are calculated. Visits after the insertion require an update of $Wait$, *Arrival*, *Start* and $MaxDelay$. We use the following formulas to update the visits after the insertion position, when visit j is inserted between i and k:

$$\begin{aligned}
Wait_k &= \max[0, Wait_k - TimeInsertion_j] \\
Arrival_k &= Arrival_k + TimeInsertion_j \\
TimeInsertion_k &= \max[0, TimeInsertion_j - Wait_k] \\
Start_k &= Start_k + TimeInsertion_k \\
MaxDelay_k &= MaxDelay_k - TimeInsertion_k
\end{aligned}$$

The formulas are then used to update the visits after visit k, until $TimeInsertion$ is reduced to zero. Visits before visit j may require an update of $MaxDelay$, making use of formula (11).

for *each non included visit* **do**
  Calculate $TimeInsertion$ for every position;
  Take position with smallest $TimeInsertion$;
  Check feasibility;
  Calculate $Ratio$;
end
Insert visit with highest ratio;
Visit j: calculate $Arrival$, $Start$, $Wait$;
for *each visit after j (until* $TimeInsertion$ *= 0)* **do**
  Update $Arrival$, $Start$, $Wait$, $MaxDelay$, $TimeInsertion$;
end
Visit j: update $MaxDelay$;
for *each visit before j* **do**
  Update $MaxDelay$;
end

Algorithm 1: Insertion step

Now we discuss the example to illustrate the insertion step. We consider a tour that consists of three locations, where the first and the last location is location 0. The time budget for this tour equals 30. Visit 1 is already inserted. Since the arrival of visit 1 is before the start of the time window, the waiting time equals 7. The values for $Arrival$, $Start$, $Wait$ and $MaxDelay$ are presented in Table 3.

Table 3: Tour 0-1-0

Location	Arrival	Start	Wait	MaxDelay
0	0	0	0	8
1	3	10	7	8
0	22	22	0	8

Location 2 and 3 are not included so we want to find the best visit to insert and its position. We calculate $TimeInsertion$ for both visits and positions and we check if insertion is feasible. We calculate the ratio for the smallest $TimeInsertion$ of feasible insertions:

Firstly, we calculate $TimeInsertion$ and $Ratio$ for visit 2:

At position 1:

$$\begin{aligned} TimeInsertion_2 = c_{02} + Wait_2 + T_2 + c_{21} - c_{01} &= 4 + 0 + 4 + 5 - 3 = 10 \\ &\leq Wait_1 + MaxDelay_1 = 7 + 10 = 17 \end{aligned}$$

At position 2:

$$\begin{aligned} TimeInsertion_2 = c_{12} + Wait_2 + T_2 + c_{20} - c_{01} &= 5 + 0 + 4 + 4 - 3 = 10 \\ &\leq Wait_0 + MaxDelay_0 = 0 + 10 = 10 \end{aligned}$$

Because $TimeInsertion$ of both positions is equal, it makes no difference, so we take position 1 and calculate $Ratio_2$:

$$Ratio_2 = \frac{S_2^2}{TimeInsertion_2} = \frac{4^2}{10} = 1.6$$

Secondly, we calculate $TimeInsertion$ and $Ratio$ for visit 3:

At position 1:

$$\begin{aligned} TimeInsertion_3 = c_{03} + Wait_3 + T_3 + c_{31} - c_{01} &= 5 + 0 + 6 + 4 - 3 = 12 \\ &\leq Wait_1 + MaxDelay_1 = 7 + 10 = 17 \end{aligned}$$

At position 2:

$$\begin{aligned} TimeInsertion_3 = c_{13} + Wait_3 + T_3 + c_{30} - c_{01} &= 4 + 0 + 6 + 5 - 3 = 12 \\ &\geq Wait_0 + MaxDelay_0 = 0 + 10 = 10 \end{aligned}$$

Because an insertion of visit j at position 2 is not feasible, we take position 1 and calculate $Ratio_3$:

$$Ratio_3 = \frac{S_3^2}{TimeInsertion_3} = \frac{8^2}{12} = 5.3$$

We see $Ratio_3$ is larger than $Ratio_2$, so we insert visit 3 at position 1. The arrival at visit 3 is 5. The start of the time window is 5, which means the start of the service is equal to 5 and there is no waiting time. Now $Arrival$, $Start$, $Wait$, $MaxDelay$ and $TimeInsertion$ of each visit after visit 3 need to be updated applying the formulas mentioned above:

$$\begin{gathered} Wait_1 = \max[0, Wait_1 - TimeInsertion_3] = \max[0, 7 - 12] = 0 \\ Arrival_1 = Arrival_1 + TimeInsertion_3 = 3 + 12 = 15 \\ TimeInsertion_1 = \max[0, TimeInsertion_3 - Wait_1] = \max[0, 12 - 7] = 5 \\ Start_1 = Start_1 + TimeInsertion_1 = 10 + 5 = 15 \\ MaxDelay_1 = MaxDelay_1 - TimeInsertion_1 = 10 - 5 = 5 \end{gathered}$$

$$Wait_0 = \max[0, Wait_0 - TimeInsertion_1] = \max[0, 0-5] = 0$$
$$Arrival_0 = Arrival_0 + TimeInsertion_1 = 20 + 5 = 25$$
$$TimeInsertion_0 = \max[0, TimeInsertion_1 - Wait_0] = \max[0, 5-0] = 5$$
$$Start_0 = Start_0 + TimeInsertion_0 = 20 + 5 = 25$$
$$MaxDelay_0 = MaxDelay_0 - TimeInsertion_0 = 10 - 5 = 5$$

For the visit after visit 3, visit 0, $MaxDelay$ needs to be updated:

$$MaxDelay_0 = \min[C_0 - Start_0, Wait_3 + MaxDelay_3] = \min[30-0, 0+5] = 5$$

The updated values for $Arrival$, $Start$, $Wait$ and $MaxDelay$ are presented in Table 4.

Table 4: Tour 0-3-1-0 (after the insertion step)

Location	Arrival	Start	Wait	MaxDelay
0	0	0	0	5
3	5	5	0	5
1	15	15	0	5
0	25	25	0	5

4.3 Shake step

In the shake step the algorithm tries to escape from a local optimum by removing a number of visits in each route. The shake step takes as input two integers: (a) the number of consecutive visits to remove from each tour ($RemoveNumber_d$) and (b) the place in the tour to start the removing process ($StartNumber_d$). If throughout the removal process, the end location is reached, then the removal continues with the visits following the start location. During the execution of the algorithm, the value of $StartNumber_d$ will become different for different tours, due to different tour lengths. This increases the possibility to escape from local optima.

After the visits are removed, all visits visited after the removed visits are shifted towards the beginning of the tour. This is to avoid unnecessary waiting. If a visit starts at the beginning of its time window, that visit and the visits following that visit remain unchanged. The shifted visits are updated using the same process as used for the insertion step. For the visits before the removed visits, only $MaxDelay$ is updated.

for *each tour* **do**
  Delete the set of visits (i to j);
  Calculate extra time available;
  for *each visit after j (until* $TimeInsertion$ *= 0)* **do**
    Shift visit towards the beginning of the tour;
    Update $Arrival$, $Start$, $Wait$, $MaxDelay$, $TimeInsertion$;
  end
  for *each visit before* i **do**
    Update $MaxDelay$;
  end
end

Algorithm 2: Shake step

To illustrate the shake step, we apply this step on the previous example with $StartNumber$ equal to 2 and $RemoveNumber$ equal to 1. We do not count visit 0 (the starting point of the tour) because the starting point needs to be location 0. This means a $StartNumber$ equal to 2 results in a removal of visit 1.

$$TimeInsertion_3 = c_{30} - c_{31} - c_{10} - T_1 - Wait_1 = 5 - 4 - 3 - 7 - 0 = -9$$
$$Arrival_0 = Arrival_0 + TimeInsertion_3 = 25 - 9 = 16$$
$$Start_0 = \max[O_0, Arrival_0] = \max[0, 16] = 16$$
$$Wait_0 = \max[0, Start_0 - Arrival_0] = \max[0, 16 - 16] = 0$$
$$TimeInsertion_0 = \min[0, TimeInsertion_3 + Wait_1] = \min[0, -9 + 0] = -9$$
$$MaxDelay_0 = MaxDelay_0 - TimeInsertion_0 = 5 + 9 = 14$$

Now we update $MaxDelay$ of visit 3 and visit 0:

$$MaxDelay_3 = \min[C_3 - Start_3, Wait_0 + MaxDelay_0] = \min[20 - 5, 0 + 14] = 14$$
$$MaxDelay_0 = \min[C_0 - Start_0, Wait_3 + MaxDelay_3] = \min[30 - 0, 0 + 14] = 14$$

The updated values for $Arrival$, $Start$, $Wait$ and $MaxDelay$ are presented in Table 5.

Table 5: Tour 0-3-0 (after the shake step)

Location	Arrival	Start	Wait	MaxDelay
0	0	0	0	14
3	5	5	0	14
0	16	16	0	14

4.4 Iterated local search heuristic

Algorithm 3 presents the iterated local search heuristic. The heuristic starts with a set of empty tours and initializes the $StartNumber$ and $RemoveNumber$ of the shake step to one. The heuristic starts executing the insertion step, until no other visits can be added to a tour. If this solution is better than the best found solution so far, i.e. if the score is higher, than the solution is recorded and $RemoveNumber$ is reset to one for the shake step. If this solution is not better, $NumberOfNoTimesNoImprovements$ is increased by one. Now the shake step is applied. After each shake step, $StartNumber$ is increased by the value of $RemoveNumber$ and $RemoveNumber$ is increased by one for the next shake step. If $StartNumber$ is equal or greater than the size of the smallest tour, the $StartNumber$ for the next shake step is decreased by the size of the smallest tour. $RemoveNumber$ is reset to one if it equals the number of locations divided by three times the number of tours.

```
startNumber = 1 ;
removeNumber = 1 ;
NumberOfTimesNoImprovement = 0 ;
while NumberOfTimesNoImprovement < 150 do
    while insertions possible do
        Insert;
    end
    if Solution better than BestFound then
        BestFound = Solution;
        removeNumber = 1 ;
        NumberOfTimesNoImprovement = 0 ;
    else
        NumberOfTimesNoImprovement =
        NumberOfTimesNoImprovement +1;
    end
    Shake Solution(removeNumber, startNumber);
    startNumber = startNumber + removeNumber;
    removeNumber = removeNumber + 1 ;
    if  startNumber >= Size of Smallest Tour then
        startNumber = startNumber - Size of Smallest Tour;
    end
    if removeNumber == n/(3*m) then
        removeNumber = 1 ;
    end
end
Return BestFound;
```

Algorithm 3: Iterated Local Search

5 Experimental results

In this chapter, we present our results. Firstly, we explain the test instances used to test the heuristic. Secondly, we compare our results with the results of Vansteenwegen et al. (2009).

5.1 Test instances

The test instances used by Vansteenwegen et al. (2009) are used to test the heuristic and the results are compared. Vansteenwegen et al. used the test instances of Righini and Salani (2006) and Montemanni and Gambardella (2009). Righini and Salani designed 58 instances for the OPTW using data set of Solomon (1987) of vehicle routing problems with time windows (c*_100, r*_100 and rc*_100) and 10 multi-depot vehicle routing problems of Cordeau et al. (1987) (pr1-pr10). Montemanni and Gambardella added 27 extra instances based on Solomon (c*_200, r*_200 and rc*_200) and 10 instances based on Cordeau et al. (pr11-pr20). All Solomon instances have 100 possible visits. The number of possible visits of the Cordeau et al. instances varies between 48 and 288. All test instances, with the number of tours varying from 1 to 4, are used to test the program and to compare the results.

5.2 Results

All computations were carried out on a personal computer Intel core i5 with 2.6 GHz and 6 GB Ram. We used Matlab 2013a to program the heuristic. Tables A.1 - A.8 of the appendix give a detailed comparison of the results obtained by this program by the results obtained by Vansteenwegen et al. (2009). The program of Vansteenwegen et al. is denoted by VSW and our program is denoted by BUI. The first column gives the instance's name. The second column gives the score obtained by Vansteenwegen et al. and the third column presents the score obtained by our program. In column four, the gap between the solution of Vansteenwegen et al. and the solution found is given, stated as a percentage of the score of Vansteenwegen et al. In the fifth column, the number of visited locations of the solution is presented. Column six gives the computation time in seconds. For each group of problems, the average, maximum and minimum gap (in %) and computation time (in seconds) are shown. Tables A.9 - A.12 show comparisons of the scores of the programs for each number of tours.

Table 6 shows a comparison of the scores of both programs, for all number of tours together. A positive sign means the program of Vansteenwegen et al. gives a higher score and vice versa. The average gap between the score of Vansteenwegen et al. and the score found is 1.2 %. In the worst case the gap is 11.1 % and in the best case -7,3 %. In 168 cases the solution of Vansteenwegen et al. is better, but in 53 cases our solution is better. In 83 cases the scores are equal. An explanation for differences in scores is a different interpretation of some parts of the heuristic. For instance, for the insertion step Vansteenwegen et al. have not defined which position to choose if two or more positions have equal values for $TimeInsertion$. Furthermore, they have not defined which location to insert if two or more locations have equal values for $Ratio$.

Table 6: Differences in scores of the program of Vansteenwegen et al. and our program

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	2.3	11.1	-2.9	80	5	31
Solomon 200	0.5	4.7	-2.8	45	16	47
Cordeau 1-10	0.6	4.7	-3.6	21	14	5
Cordeau 11-20	0.5	8.2	-7.3	22	18	0
All	1.2	11.1	-7.3	168	53	83

Table 7 shows the average computation time for the programs. For both programs the table shows the average computation time per set of test instances and per number of tours. Based on Table 7, it can be concluded that the computation time is positively correlated with the number of tours.

The program of Vansteenwegen et al. is faster. The differences in computation times can be explained by making use of different programming languages, carrying out computations on different computers and differences in implementation of the heuristic as mentioned above.

Our program is still many times faster than other methods that solve this problem, such as the ant colony system of Montemanni and Gambardella (2009) that require more than 200 seconds for the easiest problems with only one tour and the fastest method of Righini and Salani (2006) that require more than 400 seconds for the easiest problems with only one tour.

Table 7: Average CPU time for the program of Vansteenwegen et al. (VSW) and our program (BUI) (s)

Program	VSW	BUI	VSW	BUI	VSW	BUI	VSW	BUI
m	1		2		3		4	
Solomon 100	0.2	2.1	0.9	7.8	1.5	10.4	2.4	15.8
Solomon 200	1.7	13.3	2.6	29.6	1.7	22.5	1.0	11.9
Cordeau 1-10	1.8	15.4	4.8	60.1	9.2	81.5	14.1	132.7
Cordeau 11-20	2.0	18.6	5.2	58.9	9.7	118.0	13.7	127.4
All	1.2	10.0	2.6	29.2	3.7	38.2	4.9	44.5

6 MCTOPTW

The only constraint in the TOPTW of Vansteenwegen et al. (2009) is the time available. In reality, tourists need to take into account more restrictions, such as a limited money budget. The Multi Constrained Team Orienteering Problem with Time Windows (MCTOPTW) is based on the TOPTW, but with the possibility to take more constraints into account. The MCTOPTW consists of a set of locations, each of them with a certain score, a time window and one or more associated attributes, such as an entrance fee. Garcia et al. (2010) proposed an algorithm based on the algorithm of Vansteenwegen et al. (2009) to tackle the MCTOPTW. In this chapter, we discuss the adjustments made to the heuristic of Vansteenwegen et al. (2009). We explain the test instances used to test the adjusted heuristic and we compare our results with the results of Garcia et al. (2010).

6.1 Adjustments to the heuristic

In order to solve the MCTOPTW, we adjust algorithm 3 following Garcia et al. (2010). Firstly, we change the feasibility check of insertions. For the TOPTW only the time feasibility was checked. The insertion of extra attribute constraints requires checking each of the constraints to check the feasibility of an insertion of a visit. For each non included visit, we first inspect each constraint feasibility before checking the time feasibility, because the time check is computationally more expensive.

Moreover, the ratio function that determines which visit is the best one to insert, needs to be changed. For the TOPTW, the ratio only takes into account the score of the location and the time required to insert the visit. This ratio is not optimal to use in the MCTOPTW, since the attribute constraints are not taken into account. Garcia et al. analyzed different possibilities in order to define the best ratio function. Empirical tests indicated that the following ratio function is the best:

$$Ratio_i = \frac{S_i^2}{\frac{TimeInsertion_i}{availableTime} + \sum_{k=1}^{l} \frac{1}{K} \frac{e_{ik}}{available_k}} \tag{15}$$

In this ratio function, e_{ik} is the value related to attribute constraint k associated to location i. For each constraint, the value is divided by the available quantity of that constraint. The optimal weight for the attribute constraints is obtained by setting the weight of each constraint as the inverse of the number of constraints (e.g. 0.5 for two attribute constraints).

With this weighting the insertion time is equally important as the attribute constraints together. More attribute constraints will not increase the total weight of the denominator. Moreover, in this ratio function the quantity that is still available for each constraints is important and more relevant than the upper bound of the constraint.

The third change, is the change in maximum number of iterations without improvements. In the heuristic a standard number of 150 is used. Garcia et al. made this number problem dependent. To do this, we take the size of the first route of the initial solution as a indication of the number of locations that can be visited per route and the degree of difficulty of the problem. Therefore, the maximum number of iterations without improvement ($MaxIter$) is:

$$MaxIter = FactorNoImprovement * SizeOfFirstRoute \tag{16}$$

The parameter $FactorNoImprovement$ needs to be predefined. Preliminary tests showed that changing this parameter did not significantly improve the results and only caused longer computation times. Therefore, $FactorNoIprovement$ is set equal to 10 and the maximum number of iterations without improvement is:

$$MaxIter = 10 * SizeOfFirstRoute \tag{17}$$

The heuristic shown in algorithm 3 with the adjustments mentioned above is used to solve the MCTOPTW.

6.2 Test instances

Since no test problems for the MCTOPTW were available in the literature, Garcia et al. designed a new test based on the available test sets for the TOPTW. Although it is possible to add more attribute constraints, they tested their heuristic with two attribute constraints added to the test instances of Solomon and Cordeau et al. Therefore, they added two attribute values (e_{i1} and e_{i2}) to each location in the data sets. The maximum value E_1 and E_2 of each optimal solution has been calculated based on the values associated to the visited locations. These values are assigned as maximum values for the attribute constraints. The new test instances are used for problems with the number of constraints and the number of tours equal to one and two.

6.3 Results

Tables A.13 - A.16 of the appendix give a detailed comparison of the results obtained by this program by the results obtained by Garcia et al. The program of Garcia et al. is denoted by GAR and our program is denoted by BUI. The first column gives the instance's name. The following group of columns gives information about the program with one attribute constraint: The second column gives the score obtained by Garcia et al. and the third column presents the score obtained by our program. In column four, the gap between the solution of Garcia et al. and the solution found is given, stated as a percentage of the score of Garcia et al. In the fifth column, the number of visited locations of our solution is presented. Column six gives the computation time in seconds. The following group of columns gives the same information for the program with two attribute constraints. For each group of problems, the average, maximum and minimum gap (in %) and computation time (in seconds) are shown. Tables A.17 - A.20 show comparisons of the scores of the programs for each number of tours and constraints.

Table 8 shows a comparison of the scores of both programs, for all number of tours and number of constraints together. A positive sign means the program of Garcia et al. gives a higher score and vice versa. The average gap between the score of Garcia et al. and the score found is -0.1 %. This means our program gives better results on average. In the worst case the gap is 11.6 % and in the best case -9,4 %. In 38 cases the solution of Garcia et al. is better, but in 39 cases our solution is better. In 30 cases the scores are equal. Explanation for differences in scores is a different interpretation of some parts of the heuristic of Vansteenwegen et al. as mentioned above. A different interpretation of the adjustments proposed by Garcia et al. is less likely, since their article leaves no space to interpret it differently.

Table 8: Differences in scores of the program of Garcia et al. and our program

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	0.6	11.6	-9.4	35	15	37
Cordeau 1-10	-2.0	2.2	-9.2	3	15	2
All	-0.1	11.6	-9.4	38	30	39

Table 9 shows the average computation time for the programs. For both programs the table shows the average computation time per set of test instances, per number of tours and per number of constraints. This table also shows a positive correlation between the computation time and the number of tours, but the computation times are slightly lower for two constraints than for one constraint.

The program of Garcia et al. is faster. Again, the differences in computation times can be explained by making use of different programming languages, carrying out computations on different computers and differences in implementation of the heuristic. The computation times are lower than the computations times of the program solving the TOPTW for the number of tours equal to one and two. An explanation for this could be that checking the feasibility of an insertion in the MCTOPTW takes less time, because checking the feasibility of the two attribute constraints is computationally less expensive than checking the time feasibility.

Table 9: Average CPU time for the program of Garcia et al. (GAR) and our program (BUI) (s)

Program	GAR	BUI	GAR	BUI	GAR	BUI	GAR	BUI
m	1				2			
Constraints	1		2		1		2	
Solomon 100	0.3	1.5	0.2	1.0	1.2	2.2	1.2	2.0
Cordeau 1-10	4.3	22.1	3.8	17.1	15	67.1	12.4	58.8
All	1.3	6.8	1.1	5.1	4.7	18.8	4.1	16.6

7 Conclusions and future research

In this chapter we discuss our conclusions and possible topics of further research.

In this thesis the Team Orienteering Problem with Time Windows (TOPTW) is discussed. In the TOPTW, a set of locations is given, each with a score, a service time and a time window. The goal is to maximize the sum of the collected scores by a fixed number of routes, while the visits are within the time windows of the locations and the time budget of the tourist.

We discussed the algorithm used to solve the TOPTW developed by Vansteenwegen et al. (2009). They developed an iterated local search heuristic with an insertion step and a shake step. We implemented this heuristic and reproduced their results. The average gap between the score of Vansteenwegen et al. and the score found is 1.2 %, which could be explained by a different interpretation of some parts of the heuristic. Moreover, the program of Vansteenwegen et al. is faster. The differences in computation times can be explained by making use of different programming languages, carrying out computations on different computers and differences in the implementation of the heuristic. Our program is still many times faster than other methods that solve this problem, such as the ant colony system of Montemanni and Gambardella (2009) and the best program of Righini and Salani (2006).

In reality, tourists need to take into account several restrictions. The only constraint in the TOPTW of Vansteenwegen et al. is the time available. Therefore, we extended the problem by adding the possibility to add more constraints, such as a limited money budget. This problem can be solved as a Multi Constrained Team Orienteering Problem with Time Windows (MC-TOPTW). We adjusted the TOPTW heuristic following Garcia et al. (2010). We changed the feasibility check, the ratio function and the maximum number of iterations. We implemented the adjusted heuristic and reproduced the results of Garcia et al. It turned out that the average gap between the score of Garcia et al. and the score found is -0.1 %. Explanation for differences in scores is a different interpretation of some parts of the heuristic of Vansteenwegen et al. or a different interpretation of the adjustments to the heuristic of Garcia et al.

The program of Garcia et al. is faster. Again, the differences in computation times can be explained by making use of different programming languages, carrying out computations on different computers and differences in the implementation of the heuristic. The computation times of the program solving the MCTOPTW are lower than the computations times of the program solving the TOPTW. An explanation for this could be that checking the feasibility of an insertion in the MCTOPTW takes less time, because checking the feasibility of the two attribute constraints is computationally less expensive than checking the time feasibility.

A topic of further research is to investigate new possible actions such as insertion two or more visits simultaneously in an insertion step or move visits between tours. Regarding the MCTOPTW, a topic of further research is to develop a new heuristic to tackle MCTOPTW problems to compare with the one discussed in this thesis. Furthermore, new test instances can be made to test the performance of the heuristic.

References

[1] Garcia A., Vansteenwegen P., Souffriau W., Arbelaitz O. & Linaza M.T. (2010) Solving multi constrained team orienteering problems to generate tourist routes. *Centre for Industrial Management, Katholieke Universiteit Leuven*, Leuven, Belgium.

[2] Kantor M. & Rosenwein M. (1992) The orienteering problem with time windows. *The Journal of the Operational Research Society*. 43(6). p. 629-635.

[3] Labadi N., Mansini R., Melechovsk J. & Wolfler Calvo R. (2012) The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research*. 220. p. 15-27.

[4] Lin S.W. & Yu V.F. (2012) A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*. 217(1). p. 94-107.

[5] Montemanni R. & Gambardella L.M. (2009) An ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*. 34(4). p. 287-306.

[6] Righini G. & Salani M. (2006) Dynamic programming for the orienteering problem with time windows. *Dipartimento di Tecnologie dell Informazione, Universita degli Studi Milano*, Crema, Italy.

[7] Solomon M. (1987) Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*. 35. p. 254-265.

[8] Vansteenwegen P., Souffriau W., Vanden Berghe G. & Van Oudheusden D. (2009) Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*. 36. p. 3281-3290.

[9] Vansteenwegen P., Souffriau W. & Van Oudheusden D. (2011) The orienteering problem: A survey. *European Journal of Operational Research*. 209(1). p. 1-10.

A Appendix

Table A.1: Results for Solomon's test problems (m=1)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
c101	320	310	3.1	9	1.7	c201	849	840	1.1	27	8.0
c102	360	360	0.0	11	2.2	c202	910	900	1.1	30	10.0
c103	390	380	2.6	10	2.0	c203	940	920	2.1	31	9.2
c104	400	390	2.5	10	2.1	c204	950	960	-1.1	32	13.3
c105	340	340	0.0	10	2.0	c205	900	900	0.0	30	8.5
c106	340	340	0.0	10	2.0	c206	910	910	0.0	30	9.0
c107	360	350	2.8	10	2.1	c207	910	900	1.1	29	8.5
c108	370	360	2.7	11	2.3	c208	930	930	0.0	31	10.5
c109	380	380	0.0	11	2.2						
r101	182	182	0.0	7	1.2	r201	788	781	0.9	36	10.6
r102	286	286	0.0	11	2.3	r202	880	904	-2.7	46	16.4
r103	286	286	0.0	10	2.0	r203	980	956	2.4	50	12.5
r104	297	297	0.0	11	2.3	r204	1073	1045	2.6	53	14.0
r105	247	240	2.8	10	2.1	r205	931	932	-0.1	44	13.3
r106	293	293	0.0	11	2.2	r206	996	978	1.8	47	8.2
r107	288	286	0.7	10	2.1	r207	1038	1018	1.9	50	15.2
r108	297	297	0.0	11	2.4	r208	1069	1075	-0.6	55	15.0
r109	276	276	0.0	11	2.9	r209	926	917	1.0	46	11.8
r110	281	281	0.0	11	2.4	r210	958	949	0.9	49	17.4
r111	295	294	0.3	12	3.1	r211	1023	995	2.7	48	19.3
r112	295	290	1.7	11	2.3						
rc101	219	203	7.3	9	1.6	rc201	780	779	0.1	35	9.9
rc102	259	232	10.4	9	1.8	rc202	882	891	-1.0	38	16.5
rc103	265	265	0.0	11	1.6	rc203	960	948	1.3	40	11.1
rc104	297	297	0.0	11	1.7	rc204	1117	1118	-0.1	47	24.6
rc105	221	215	2.7	10	2.1	rc205	840	835	0.6	37	11.7
rc106	239	239	0.0	11	2.2	rc206	860	864	-0.5	36	16.3
rc107	274	274	0.0	11	2.6	rc207	926	929	-0.3	41	23.7
rc108	288	283	1.7	11	2.4	rc208	1037	1000	3.6	40	13.8
Average			1.4		2.1	Average			0.6		13.3
Max			10.4		3.1	Max			3.6		24.6
Min			0.0		1.2	Min			-2.7		8.0

Table A.2: Results for Solomon's test problems (m=2)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
c101	590	570	3.4	19	6.5	c201	1400	1400	0.0	59	29.4
c102	650	650	0.0	21	8.6	c202	1430	1410	1.4	60	25.1
c103	700	690	1.4	20	9.6	c203	1430	1440	-0.7	64	27.4
c104	750	750	0.0	22	12.9	c204	1460	1450	0.7	64	47.1
c105	640	640	0.0	21	7.5	c205	1450	1440	0.7	63	34.2
c106	620	620	0.0	20	7.2	c206	1440	1450	-0.7	64	25.7
c107	670	670	0.0	22	8.6	c207	1450	1460	-0.7	65	49.2
c108	670	670	0.0	22	6.8	c208	1460	1470	-0.7	66	28.4
c109	710	700	1.4	22	8.5						
r101	330	330	0.0	13	4.1	r201	1231	1223	0.6	71	58.1
r102	508	508	0.0	21	8.5	r202	1270	1305	-2.8	80	49.3
r103	513	506	1.4	20	6.3	r203	1377	1354	1.7	83	32.1
r104	539	537	0.4	22	8.8	r204	1440	1427	0.9	93	21.2
r105	430	412	4.2	17	5.0	r205	1338	1315	1.7	81	18.0
r106	529	529	0.0	21	7.9	r206	1401	1377	1.7	85	41.7
r107	529	520	1.7	20	7.8	r207	1428	1428	0.0	93	31.7
r108	549	545	0.7	22	10.0	r208	1458	1450	0.5	97	21.5
r109	498	491	1.4	22	8.1	r209	1345	1325	1.5	82	22.3
r110	515	498	3.3	21	10.0	r210	1365	1361	0.3	85	23.4
r111	535	526	1.7	23	8.6	r211	1422	1409	0.9	91	34.5
r112	515	514	0.2	22	8.2						
rc101	427	410	4.0	19	6.5	rc201	1305	1291	1.1	61	17.3
rc102	494	461	6.7	19	6.6	rc202	1461	1392	4.7	66	18.3
rc103	519	471	9.2	17	6.2	rc203	1573	1522	3.2	83	28.3
rc104	565	533	5.7	22	7.9	rc204	1656	1628	1.7	88	16.3
rc105	459	427	7.0	20	7.1	rc205	1381	1365	1.2	69	24.7
rc106	458	440	3.9	20	6.6	rc206	1495	1466	1.9	76	25.1
rc107	515	516	-0.2	21	8.1	rc207	1531	1493	2.5	76	27.6
rc108	546	515	5.7	19	6.6	rc208	1606	1596	0.6	84	22.1
Average			2.2		7.8	Average			0.9		29.6
Max			9.2		12.9	Max			4.7		58.1
Min			-0.2		4.1	Min			-2.8		16.3

Table A.3: Results for Solomon's test problems (m=3)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
c101	790	780	1.3	29	9.9	c201	1750	1700	2.9	89	18.6
c102	890	870	2.2	30	7.7	c202	1750	1740	0.6	93	23.3
c103	960	940	2.1	33	12.6	c203	1760	1760	0.0	95	43.3
c104	1010	980	3.0	33	7.6	c204	1780	1770	0.6	96	33.3
c105	840	840	0.0	30	10.3	c205	1770	1730	2.3	92	15.2
c106	840	830	1.2	29	11.1	c206	1770	1770	0.0	96	28.1
c107	900	870	3.3	32	10.9	c207	1810	1760	2.8	95	32.9
c108	900	900	0.0	33	12.1	c208	1810	1800	0.6	99	36.3
c109	950	940	1.1	33	8.8						
r101	481	460	4.4	19	6.9	r201	1408	1402	0.4	93	19.6
r102	685	648	5.4	31	11.1	r202	1443	1443	0.0	98	19.2
r103	720	717	0.4	32	15.3	r203	1458	1458	0.0	100	18.6
r104	765	707	7.6	30	8.2	r204	1458	1458	0.0	100	7.5
r105	609	594	2.5	26	10.7	r205	1458	1458	0.0	100	13.4
r106	719	677	5.8	29	10.6	r206	1458	1458	0.0	100	14
r107	747	693	7.2	29	11.8	r207	1458	1458	0.0	100	12.1
r108	790	702	11.1	33	11.1	r208	1458	1458	0.0	100	7.3
r109	699	670	4.1	30	13.5	r209	1458	1458	0.0	100	17.2
r110	711	679	4.5	30	12.2	r210	1458	1458	0.0	100	20
r111	764	707	7.5	30	7.7	r211	1458	1458	0.0	100	20.5
r112	758	744	1.8	32	13.4						
rc101	604	558	7.6	25	8.2	rc201	1625	1623	0.1	89	32.2
rc102	698	634	9.2	28	6.7	rc202	1686	1679	0.4	95	27.6
rc103	747	713	4.6	27	8.6	rc203	1724	1721	0.2	99	16.5
rc104	822	813	1.1	32	12.5	rc204	1724	1724	0.0	100	12.2
rc105	654	653	0.2	28	10.3	rc205	1659	1672	-0.8	94	33.1
rc106	678	668	1.5	28	9.9	rc206	1708	1712	-0.2	98	20.6
rc107	745	745	0.0	31	11.1	rc207	1713	1724	-0.6	100	46.4
rc108	757	740	2.2	28	10.3	rc208	1724	1724	0.0	100	18.7
Average			3.5		10.4	Average			0.3		22.5
Max			11.1		15.3	Max			2.9		46.4
Min			0.0		6.7	Min			-0.8		7.3

Table A.4: Results for Solomon's test problems (m=4)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
c101	1000	970	3.0	39	14.6	c201	1810	1810	0	100	15.1
c102	1090	1090	0.0	43	19.3	c202	1810	1810	0	100	13.4
c103	1150	1160	-0.9	42	17.9	c203	1810	1810	0	100	13.4
c104	1220	1200	1.6	44	18.1	c204	1810	1810	0	100	8.6
c105	1030	1040	-1.0	41	16.2	c205	1810	1810	0	100	13.3
c106	1040	1070	-2.9	42	18.3	c206	1810	1810	0	100	3.6
c107	1100	1100	0.0	44	22.7	c207	1810	1810	0	100	10.7
c108	1100	1060	3.6	40	14.6	c208	1810	1810	0	100	9.5
c109	1180	1140	3.4	44	10.6						
r101	601	590	1.8	28	10.2	r201	1458	1458	0	100	15.4
r102	807	794	1.6	36	11.1	r202	1458	1458	0	100	13.3
r103	878	850	3.2	40	18.6	r203	1458	1458	0	100	9.6
r104	941	881	6.4	40	13.5	r204	1458	1458	0	100	7.5
r105	735	742	-1.0	35	12.8	r205	1458	1458	0	100	10.5
r106	870	831	4.5	39	15.2	r206	1458	1458	0	100	13.3
r107	927	894	3.6	42	19.2	r207	1458	1458	0	100	8.1
r108	982	954	2.9	45	22.5	r208	1458	1458	0	100	5.6
r109	866	815	5.9	39	12.5	r209	1458	1458	0	100	13.7
r110	870	846	2.8	40	12.8	r210	1458	1458	0	100	15.6
r111	935	931	0.4	44	16.5	r211	1458	1458	0	100	15.2
r112	939	927	1.3	43	18.2						
rc101	794	794	0.0	37	11.3	rc201	1724	1724	0	100	21.5
rc102	881	845	4.1	38	11.1	rc202	1724	1724	0	100	15.9
rc103	947	940	0.7	42	21	rc203	1724	1724	0	100	3.3
rc104	1019	996	2.3	40	17.3	rc204	1724	1724	0	100	1.5
rc105	841	806	4.2	39	15.3	rc205	1724	1724	0	100	21.0
rc106	874	850	2.7	37	11.8	rc206	1724	1724	0	100	14.2
rc107	951	941	1.1	40	18.5	rc207	1724	1724	0	100	13.9
rc108	998	933	6.5	39	15.6	rc208	1724	1724	0	100	15.3
Average			2.1		15.8	Average			0.0		11.9
Max			6.5		22.7	Max			0.0		21.5
Min			-2.9		10.2	Min			0.0		1.5

Table A.5: Results for the test problems of Cordeau, Gendreau and Laporte (m=1)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
pr01	304	304	0.0	20	2.3	pr11	330	325	1.5	20	1.7
pr02	385	382	0.8	20	4.9	pr12	431	422	2.1	23	4.7
pr03	384	388	-1.0	21	8.4	pr13	450	434	3.6	25	12.9
pr04	447	449	-0.4	22	14.0	pr14	482	517	-7.3	27	18.9
pr05	576	565	1.9	31	21.0	pr15	638	656	-2.8	36	17.9
pr06	538	528	1.9	26	28.9	pr16	559	568	-1.6	30	34.3
pr07	291	291	0.0	16	6.7	pr17	346	355	-2.6	18	3.8
pr08	463	463	0.0	25	10.5	pr18	479	455	5.0	23	16.3
pr09	461	468	-1.5	25	27.7	pr19	499	462	7.4	27	26.4
pr10	539	543	-0.7	29	29.9	pr20	570	599	-5.1	34	48.7
Average			0.1		15.4	Average			0.0		18.6
Max			1.9		29.9	Max			7.4		48.7
Min			-1.5		2.3	Min			-7.3		1.7

Table A.6: Results for the test problems of Cordeau, Gendreau and Laporte (m=2)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
pr01	471	450	4.5	30	3.6	pr11	542	540	0.4	36	6.3
pr02	660	667	-1.1	39	16.4	pr12	727	724	0.4	40	12.3
pr03	714	717	-0.4	39	33.9	pr13	757	777	-2.6	45	32.7
pr04	863	857	0.7	45	57.3	pr14	925	874	5.5	49	74.8
pr05	1011	1020	-0.9	56	109.2	pr15	1126	1107	1.7	61	78.8
pr06	997	996	0.1	52	82.3	pr16	1110	1098	1.1	57	156.9
pr07	552	544	1.4	32	6.2	pr17	624	630	-1.0	37	13.3
pr08	796	777	2.4	42	48.5	pr18	877	872	0.6	46	38.4
pr09	867	826	4.7	50	46.3	pr19	955	877	8.2	52	64.2
pr10	1004	1040	-3.6	56	197.1	pr20	1056	1098	-4.0	59	110.8
Average			0.8		60.1	Average			1.0		58.8
Max			4.7		197.1	Max			8.2		156.9
Min			-3.6		3.6	Min			-4.0		6.3

Table A.7: Results for the test problems of Cordeau, Gendreau and Laporte (m=3)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
pr01	598	598	0.0	42	3.5	pr11	632	635	-0.5	44	4.2
pr02	899	906	-0.8	57	46.1	pr12	902	912	-1.1	60	24.3
pr03	946	935	1.2	56	43.4	pr13	1046	1099	-5.1	68	47.3
pr04	1195	1195	0.0	70	86.3	pr14	1197	1226	-2.4	71	149.1
pr05	1356	1350	0.4	73	136.0	pr15	1488	1495	-0.5	89	144.3
pr06	1376	1371	0.4	77	141.3	pr16	1478	1512	-2.3	83	230.0
pr07	713	709	0.6	45	25.2	pr17	808	803	0.6	52	29.0
pr08	1082	1054	2.6	58	39.0	pr18	1165	1182	-1.5	70	86.2
pr09	1144	1117	2.4	69	97.3	pr19	1238	1322	-6.8	77	152.0
pr10	1473	1440	2.2	79	197.2	pr20	1514	1507	0.5	84	313.7
Average			0.9		81.5	Average			-1.9		118.0
Max			2.6		197.2	Max			0.6		313.7
Min			-0.8		3.5	Min			-6.8		4.2

Table A.8: Results for the test problems of Cordeau, Gendreau and Laporte (m=4)

Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	Name	VSW	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
pr01	644	649	-0.8	46	5.3	pr11	657	654	0.5	47	3.0
pr02	1014	1007	0.7	65	28.6	pr12	1118	1041	6.9	66	23.2
pr03	1162	1146	1.4	74	122.6	pr13	1329	1273	4.2	89	71.2
pr04	1452	1463	-0.8	84	194.4	pr14	1568	1502	4.2	92	196.5
pr05	1665	1640	1.5	95	247.2	pr15	1854	1814	2.2	104	177.6
pr06	1696	1653	2.5	91	221.8	pr16	1887	1877	0.5	105	272.5
pr07	840	805	4.2	56	10.6	pr17	925	877	5.2	61	17.3
pr08	1267	1269	-0.2	72	55.1	pr18	1470	1381	6.1	82	78.8
pr09	1460	1465	-0.3	92	126.1	pr19	1596	1598	-0.1	98	184.7
pr10	1782	1802	-1.1	103	315.2	pr20	1841	1874	-1.8	110	249.5
Average			0.7		132.7	Average			2.8		127.4
Max			4.2		315.2	Max			6.9		272.5
Min			-1.1		5.3	Min			-1.8		3.0

Table A.9: Differences in scores (m=1)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	1.4	10.4	0	13	0	16
Solomon 200	0.6	3.6	-2.7	15	8	4
Cordeau 1-10	0.1	1.9	-1.5	3	4	3
Cordeau 11-20	0	7.4	-7.3	5	5	0

Table A.10: Differences in scores (m=2)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	2.2	9.2	-0.2	19	1	9
Solomon 200	0.9	4.7	-2.8	20	5	2
Cordeau 1-10	0.8	4.7	-3.6	6	4	0
Cordeau 11-20	1.0	8.2	-4	7	3	0

Table A.11: Differences in scores (m=3)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	3.5	11.1	0	26	0	3
Solomon 200	0.3	2.9	-0.8	10	3	14
Cordeau 1-10	0.9	2.6	-0.8	7	1	2
Cordeau 11-20	-1.9	0.6	-6.8	2	8	0

Table A.12: Differences in scores (m=4)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	2.1	6.5	-2.9	22	4	3
Solomon 200	0.0	0.0	0.0	0	0	27
Cordeau 1-10	0.7	4.2	-1.1	5	5	0
Cordeau 11-20	2.8	6.9	-1.8	8	2	0

Table A.13: Results for the MCTOPTW test problems of Solomon (m=1)

	1 constraint					2 constraints				
Name	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
c101	300	300	0.0	9	0.9	320	320	0.0	10	0.6
c102	360	360	0.0	11	2	360	360	0.0	11	1.3
c103	380	380	0.0	10	2.1	390	390	0.0	10	1.4
c104	400	390	2.5	10	2.1	400	390	2.5	10	1.5
c105	330	330	0.0	10	2.1	340	340	0.0	10	1.4
c106	340	340	0.0	10	2.1	340	330	2.9	10	1.5
c107	370	370	0.0	11	2.3	340	370	-8.8	11	1.5
c108	350	350	0.0	11	2.0	340	340	0.0	11	1.4
c109	380	380	0.0	11	1.7	370	380	-2.7	11	1.3
r101	182	182	0.0	7	0.5	182	186	-2.2	7	0.3
r102	281	283	-0.7	11	1.7	286	283	1.0	11	1.1
r103	286	286	0.0	10	1.7	286	286	0.0	10	1.1
r104	288	281	2.4	10	1.1	288	281	2.4	10	0.7
r105	247	240	2.8	10	1.6	247	247	0.0	10	0.9
r106	281	283	-0.7	11	1.3	289	283	2.1	10	0.6
r107	289	286	1.0	10	1.7	288	286	0.7	10	0.8
r108	308	308	0.0	13	2.0	295	304	-3.1	13	1.5
r109	276	276	0.0	11	2.4	277	276	0.4	11	1.9
r110	274	281	-2.6	11	2.1	277	276	0.4	10	1.5
r111	295	294	0.3	12	2.1	295	295	0.0	12	1.6
r112	292	292	0.0	11	2.1	295	297	-0.7	11	1.4
rc101	216	206	4.6	9	0.8	219	216	1.4	9	0.4
rc102	259	259	0.0	9	0.8	259	259	0.0	9	0.4
rc103	259	256	1.2	9	0.5	259	259	0.0	9	0.3
rc104	301	266	11.6	9	0.4	296	266	10.1	9	0.3
rc105	213	189	11.3	7	0.4	213	195	8.5	7	0.2
rc106	233	255	-9.4	10	1.4	227	233	-2.6	9	0.5
rc107	270	268	0.7	10	1.4	270	268	0.7	10	0.5
rc108	298	283	5.0	10	0.9	298	283	5.0	10	0.4
Average			1.0		1.5			0.6		1.0
Max			11.6		2.4			10.1		1.9
Min			-9.4		0.4			-8.8		0.2

Table A.14: Results for the MCTOPTW test problems of Solomon (m=2)

			1 constraint					2 constraints		
Name	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
c101	580	580	0.0	20	3.1	580	580	0.0	20	2.7
c102	650	650	0.0	22	3.2	650	640	1.5	22	2.8
c103	710	700	1.4	21	3.5	690	670	2.9	20	2.9
c104	760	750	1.3	21	2.7	740	740	0.0	21	2.4
c105	640	640	0.0	19	2.8	640	640	0.0	19	2.8
c106	620	620	0.0	19	2.5	620	620	0.0	19	2.4
c107	660	670	-1.5	19	2.7	670	660	1.5	19	2.6
c108	680	670	1.5	19	2.6	680	670	1.5	19	2.3
c109	710	700	1.4	21	2.8	720	700	2.8	21	2.4
r101	322	330	-2.5	13	1.6	322	341	-5.9	13	1.3
r102	508	501	1.4	17	2.6	494	501	-1.4	17	2.3
r103	512	513	-0.2	18	2.7	513	513	0.0	18	2.3
r104	538	538	0.0	19	2.3	518	531	-2.5	18	2.1
r105	434	434	0.0	19	1.9	423	423	0.0	19	1.7
r106	529	529	0.0	20	2.9	529	519	1.9	19	2.4
r107	523	523	0.0	19	1.5	527	527	0.0	19	1.5
r108	539	534	0.9	17	1.6	541	524	3.1	17	1.4
r109	498	506	-1.6	20	2.2	488	498	-2.0	19	2.0
r110	519	519	0.0	20	2.0	503	506	-0.6	19	2.0
r111	536	535	0.2	22	1.6	530	530	0.0	22	1.5
r112	513	513	0.0	19	1.8	520	522	-0.4	19	1.6
rc101	427	427	0.0	18	1.8	427	419	1.9	18	1.5
rc102	497	497	0.0	17	1.9	487	487	0.0	17	1.4
rc103	501	497	0.8	18	0.7	510	512	-0.4	19	1.5
rc104	556	551	0.9	19	1.7	551	551	0.0	19	1.6
rc105	448	438	2.2	16	1.4	448	441	1.6	16	1.4
rc106	462	464	-0.4	19	2.0	455	446	2.0	18	1.6
rc107	516	516	0.0	20	2.5	523	510	2.5	20	2.1
rc108	526	522	0.8	20	1.9	541	540	0.2	21	1.8
Average			0.2		2.2			0.3		2.0
Max			2.2		3.5			3.1		2.9
Min			-2.5		0.7			-5.9		1.3

Table A.15: Results for the MCTOPTW test problems of Cordeau, Gendreau and Laporte (m=1)

			1 constraint					2 constraints		
Name	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
pr01	290	302	-4.1	20	2.4	288	302	-4.9	20	2.3
pr02	375	385	-2.7	20	8.9	378	385	-1.9	20	8.2
pr03	380	388	-2.1	21	9.0	386	386	0.0	21	8.1
pr04	445	453	-1.8	22	19.0	455	445	2.2	21	17.0
pr05	521	569	-9.2	30	43.0	553	571	-3.3	29	32.0
pr06	534	548	-2.6	27	63.9	493	522	-5.9	26	51.6
pr07	289	288	0.3	17	3.0	289	289	0.0	17	2.9
pr08	452	458	-1.3	24	13.8	450	458	-1.8	24	11.2
pr09	461	463	-0.4	24	23.7	449	441	1.8	23	14.5
pr10	517	519	-0.4	26	34.5	502	511	-1.8	26	22.8
Average			-2.4		22.1			-1.5		17.1
Max			0.3		63.9			2.2		51.6
Min			-9.2		2.4			-5.9		2.3

Table A.16: Results for the MCTOPTW test problems of Cordeau, Gendreau and Laporte (m=2)

			1 constraint					2 constraints		
Name	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)	GAR	BUI	Gap(%)	Visits(BUI)	CPU(s)(BUI)
pr01	489	489	0.0	31	8.2	479	471	1.7	30	7.4
pr02	654	668	-2.1	39	32.6	656	667	-1.7	39	28.7
pr03	701	692	1.3	38	37.3	710	701	1.3	39	34.9
pr04	872	876	-0.5	45	60.2	846	857	-1.3	44	53.2
pr05	1002	1013	-1.1	56	105.6	1036	1026	1.0	57	98.4
pr06	952	974	-2.3	50	155.8	935	982	-5.0	50	123.6
pr07	547	558	-2.0	33	14.5	546	544	0.4	34	10.3
pr08	774	797	-3.0	42	55.7	813	803	1.2	42	55.4
pr09	828	824	0.5	50	66.4	823	826	-0.4	50	39.8
pr10	998	1014	-1.6	53	134.5	1012	1032	-2.0	54	136.3
Average			-1.1		67.1			-0.5		58.8
Max			1.3		155.8			1.7		136.3
Min			-3.0		8.2			-5.0		7.4

Table A.17: Differences in scores (m=1, 1 constraint)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	1.0	11.6	-9.4	11	4	14
Cordeau 1-10	-2.4	0.3	-9.2	1	9	0

Table A.18: Differences in scores (m=1, 2 constraints)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	0.6	10.1	-8.8	13	6	10
Cordeau 1-10	-1.5	2.2	-5.9	2	6	2

Table A.19: Differences in scores (m=2, 1 constraint)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	0.2	2.2	-2.5	11	5	13
Cordeau 1-10	-1.1	1.3	-3.0	2	7	1

Table A.20: Differences in scores (m=1, 2 constraints)

	Average gap	Max gap	Min gap	# lower	# higher	# equal
Solomon 100	0.3	3.1	-5.9	12	7	10
Cordeau 1-10	-0.5	1.7	-5.0	5	5	0