Iterated local search heuristic for the team orienteering problem with time windows

A reproduction of the original paper of Vansteenwegen et al. (2009)

Mechteld Ferment

357196

Supervisor Dr. D. Huisman

Erasmus University Rotterdam
Bachelor’s Thesis
Econometrics & Operations Research
Major Quantitative Logistics

July 2015
Abstract

Tourists have the difficult task to schedule their holiday activities in such a way that they enjoy their holiday as much as possible, while keeping the opening times and exact locations of attractions in mind. This task can be formulated as the team orienteering problem with time windows (TOPTW). The TOPTW uses a list of locations as input, together with their opening times and appreciation scores. The goal of the TOPTW is to maximize the collected appreciation scores by visiting locations, while considering the opening times and the travel times within locations.

This thesis is a replication of earlier research done by Vansteenwegen et al. [13]. Their iterated local search (ILS) heuristic is implemented by us and the results are compared and certain claims made by them are verified. Our contribution is to make adjustments to the original heuristic in order to improve the results.

Due to possible interpretation or implementation errors, the total score of our replication was 1.52% lower than the total score of Vansteenwegen et al. However, the total score of the best variant made by us is 0.28% higher than theirs, as the score of 75 test instances (out of 224) was improved. For 32 test instances the score of the best variant is higher than the best-known result until 2009.
# Table of contents

1 Introduction 3

2 Literature overview 4

3 Problem formulation 5

4 Methodology 7
   4.1 Insertion step 7
   4.2 Shake step 9
   4.3 ILS heuristic 10
   4.4 Variants of the heuristic 10
      4.4.1 Resetting the parameter R 11
      4.4.2 Resetting the parameter S 11
      4.4.3 Ratio formula 12
      4.4.4 Maximum number of iterations without improvement 12

5 Data 13
   5.1 Different datasets 13
   5.2 Time windows 13
   5.3 Optimal solutions 14

6 Results 15
   6.1 Reproduction of the ILS heuristic 15
   6.2 Resetting the parameter R 16
      6.2.1 Formulas dependent on $m$ 16
      6.2.2 Formulas independent of $m$ and the number of scheduled visits 17
      6.2.3 Formulas dependent on the number of scheduled visits 18
   6.3 Resetting the parameter S 19
   6.4 Ratio formula 20
   6.5 Maximum number of iterations without improvement 20
   6.6 Combining different variants 21
   6.7 Results of the best combination 22

7 Conclusion 24

References 25
1 Introduction

A common problem for tourists is that they have limited time to spend at their destination, while there are many appealing attractions to visit. How can they plan their days in such a way that they profit optimally from their holiday? A tourist has a certain appreciation for an attraction, indicating how much he enjoys visiting it. However, the higher appreciated attractions may not be close to each other and it is therefore not possible to visit them all during the trip. Making a feasible plan within the given time frame, while obtaining the highest possible score, is a difficult task. Furthermore, such a plan becomes infeasible or not optimal as soon as a tourist deviates from the schedule due to unexpected events.

A program that would calculate the best possible schedule within a few seconds would be a great convenience to tourists. A personalized electronic tourist guide (PET) is a device that calculates the optimal schedule, maximizing the tourist’s collected score, while taking into account the location of the attractions and their opening times, together with the travel times within the attractions. The problem solved by the PET is called the tourist trip design problem (TTDP) [12]. The team orienteering problem with time windows (TOPTW) is a simpler version of the TTDP, as factors like weather conditions and budget constraints are not taken into account. In the TOPTW a set of locations is given, together with their appreciation scores, visiting times and time windows, and used to make the best possible schedule, gaining the maximum score for a limited number of time frames. One of those time frames can be seen as one single day, where preferences of the tourist regarding departure and arrival time at the hotel can be taken into account.

An additional requirement of the PET is that it should solve the problem within seconds. Tourists need to be able to recalculate their schedule quickly when they deviate from the original plan. This can be due to unexpected closing of an attraction or a long queue, lowering the value of a visit as tourists do not wish to wait in line for a few hours during their holidays. They do not have time to or do not want to wait for a long time to recalculate their schedule but want to be able to see immediately what the new schedule suggests.

This thesis follows research conducted by Vansteenwegen et al. in their paper “Iterated local search for the team orienteering problem with time windows”, which was published in 2009 [13]. The goal is to implement their iterated local search (ILS) heuristic, which can solve the planning problem within seconds close to optimality, and to compare our results with their results. Secondly, we will try to improve their heuristic by finding a solution with a higher score, rather than finding a satisfactory solution in shorter computation time.

A literature overview is given in the next chapter and the problem formulation is given in Chapter 3. In Chapter 4 the methodology is described and the data is reviewed in Chapter 5. In Chapter 6 the results are presented and discussed, which are concluded in Chapter 7.
2 Literature overview

Much research has been done on the orienteering problem (OP) because there are many versions of the problem, which are all equivalent in a sense. For example, there is the selective traveling salesman problem [8] and the maximum collection problem [1]. Both problems optimize the collected score of visited points, while not every location can be visited due to constraints on time and distance. But any other problem which maximizes the total score of visited points, where also constraints on time and distance have to be considered, is in principle an OP. When multiple trips have to be planned within the same set of locations and a location can only be visited in one of the trips, the OP becomes the team orienteering problem (TOP).

However, time windows were often left out from the research. Naturally, the paper on which this thesis is based, Vansteenwegen et al. [13], discusses the TOPTW. In their paper, they discuss the research that has been done in and before 2009 and it is therefore omitted here. In the past few years, even more research has been done on the (T)OPTW. Lin and Vincent [9] come up with a simulated annealing heuristic, which is a local search-based heuristic that accepts, with a small probability, worse solutions in order to climb out of local optima. Labadie et al. [7] propose a variable neighborhood search algorithm which explores the neighborhoods granular instead of checking the complete neighborhoods. Labadie et al. [6] use a greedy randomized adaptive search procedure with an evolutionary local search, which is an extension of the ILS. Hu and Lim [5] suggest a method where three heuristics are combined. The first two components, the local search procedure and simulated annealing procedure, discover a set of routes. The third component recombines the routes to find the best possible solution.
3 Problem formulation

In the OPTW there are \( n \) different locations. Every location is assigned a score \( \text{Score}_i \), a visiting time \( T_i \), and a time window \([O_i, C_i]\) during which the location can be visited. The departure point at the beginning of the time frame should equal the arrival point at the end of the time frame and is often the first location in the set of locations. In the following mathematical formulation, however, the first location \((i = 1)\) is the departure point and last location \((i = n)\) is the arrival point. Time \( c_{ij} \) is the time it takes to travel from location \( i \) to location \( j \). \( T_{\text{max}} \) is the maximum available time within the time frame.

To transform this problem into the TOPTW, the OPTW should be executed \( m \) times, where the problem keeps track of which locations were visited in an earlier time frame, as executing it \( m \) separate times would give \( m \) identical schedules. This can be done by introducing a constant \( M \) and the following variables: \( x_{ijd} \), \( y_{id} \) and \( s_{id} \). \( x_{ijd} \) equals 1 if a visit of location \( i \) is followed by a visit of location \( j \) in time frame \( d \), 0 otherwise. \( y_{id} \) equals 1 if a visit of location \( i \) is included in time frame \( d \), 0 otherwise. \( s_{id} \) is the starting time of the visit of location \( i \) in time frame \( d \), which is essentially different from the arrival time at a location. It is possible to arrive at a location before the opening of the time window, but the visit can only start after waiting until the location has opened. Lastly, \( M \) should be a large constant.

The following formulation of the TOPTW is described by Vansteenwegen et al. [13]:

\[
\text{Maximize} \quad \sum_{d=1}^{m} \sum_{i=2}^{n-1} \text{Score}_i y_{id} \\
\text{Subject to} \quad \sum_{d=1}^{m} \sum_{j=2}^{n-1} x_{1jd} = \sum_{d=1}^{m} \sum_{i=2}^{n-1} x_{ind} = m \\
\sum_{i=1}^{n-1} x_{ikd} = \sum_{j=2}^{n} x_{kjd} = y_{kd} \quad (k = 2, ..., n - 1; d = 1, ..., m) \\
s_{id} + T_i + c_{ij} - s_{jd} \leq M(1 - x_{ijd}) \quad (i, j = 1, ..., n; d = 1, ..., m) \\
\sum_{d=1}^{m} y_{kd} \leq 1 \quad (k = 2, ..., n - 1) \\
\sum_{i=1}^{n-1} (T_i y_{id} + \sum_{j=2}^{n} c_{ij} x_{ijd}) \leq T_{\text{max}} \quad (d = 1, ..., m) \\
O_i \leq s_{id} \quad (i = 1, ..., n; d = 1, ..., m) \\
s_{id} \leq C_i \quad (i = 1, ..., n; d = 1, ..., m) \\
x_{ijd}, y_{id} \in [0, 1] \quad (i, j = 1, ..., n; d = 1, ..., m)
\]
The objective function (1) maximizes the total collected score of the $m$ time frames. Constraint (2) makes sure that the schedule departs at the departure location and arrives at the arrival location in each time frame. Constraints (3) make sure that only included visits in the schedule have a preceding and a subsequent visit. Constraints (4) make sure that the starting time of a visit is long enough after the starting time of the previous visit. The time in between the two starting times should be long enough to complete the visit at the first location and travel to the second one. Constraints (5) make sure that each location is visited at most once. Constraints (6) make sure that all the visiting times added up together with all the travel times within a time frame are less than $T_{max}$. Constraints (7) and (8) make sure that a visit can only start within the time window of a location. Note here that the closing time of the time window is not the actual closing of the location, but the final moment a visit can start at the location. Constraints (9) make sure that the variables $x_{ijd}$ and $y_{id}$ only take the values of 0 and 1.

An adjustment has to be made to this mathematical formulation, as constraints (6) do not include the possible waiting times at the locations. The constraints now state that all the visiting times and travel times together should be less than or equal to $T_{max}$, but it does not consider situations where the arrival at a location is before the opening of the time window. The constraints can be replaced by setting the closing time of the arrival location to $T_{max}$, as seen in constraint (10). This is enough to ensure that all the visiting times and travel times within a time frame, together with any possible waiting time, are less than or equal to $T_{max}$.

$$C_n = T_{max}$$ (10)
4 Methodology

The TOPTW problem is hard to solve to optimality due to the high amount of constraints. Golden et al. [4] have proven that the OP problem is NP-hard, such that the TOPTW, which is an extension of the OP, is also NP-hard. The TOPTW should be solved with a high quality solution in limited computation time, for which a heuristic is designed by Vansteenwegen et al. [13].

However, Gendreau et al. [3] discuss several reasons why it is hard to design an algorithm for the OP. The distance to reach a location and its score are independent and are sometimes even opposites of each other, meaning that traveling a long distance can result in visiting a location with a higher appreciation score but it obstructs the visiting of other locations due to the time constraints. This complicates the decision of which locations to include in the schedule and it makes it difficult for easy heuristics to come close to the optimal solution. Also, these easy heuristics do not have the ability to correct bad decisions made earlier satisfactorily and the time windows complicate the process even further.

Vansteenwegen et al. propose an iterated local search (ILS) heuristic, which performs well on the datasets made by Solomon [11] and Cordeau et al. [2] for the vehicle routing problem with time windows and the multi-depot vehicle routing problem, respectively. The heuristic combines an insertion step, inserting a location into the schedule, with a shake step, deleting a number of locations from the schedule in order to reschedule those parts to try to escape local optima.

The variables used in the heuristic have the same meaning as the variables used in Chapter 2. Variable $s_{id}$, indicating the starting time of a visit, is changed to $s_i$ but it does not change the meaning of the variable as a location can only be visited in a single time frame. A new variable $a_i$ is introduced to store the arrival time at a certain location separately from the starting time of a visit.

4.1 Insertion step

In the insertion step of the heuristic locations are added one by one in the schedule. Before a location can be inserted it should be checked whether the schedule is still feasible after the insertion, which is especially important due to the different time windows. As the heuristic should be solved within limited computation time, it is important to check the feasibility of the schedule in an efficient way. It takes a lot of time to check every single visit after a potential insertion and this can be avoided by adding two variables, $Wait$ and $MaxShift$, to the heuristic. With these two variables added, only the first visit after a potential insertion needs to be checked in order to verify the feasibility of the entire schedule.

The variable $Wait$ records the waiting time if the arrival at a location is prior to the opening of the time window. If the arrival is during the time window variable $Wait$ equals zero.

$$\text{Wait}_i = max[0, O_i - a_i]$$  \hspace{1cm} (11)
MaxShift is the amount of time the visit at a location can be delayed without troubling the rest of the schedule. The MaxShift of a visit is equal to the time the visit of the next location can be delayed ($MaxShift_{i+1}$) plus the time that has to be waited before the start of the next visit ($Wait_{i+1}$), unless it is restricted by the closing of the time window of the current location.

$$MaxShift_i = \min[MaxShift_{i+1} + Wait_{i+1}, C_i - s_i]$$ (12)

The total time consumption of an insertion of a visit of location $j$ in between the visits of locations $i$ and $k$ is equal to the travel time from $i$ to $j$ and $j$ to $k$, minus the travel time from $i$ to $k$, plus the possible waiting time at $j$ and the visiting time of $j$. Such an insertion is possible if the shift within the schedule caused by the insertion of $j$ is lower than or equal to the sum of $Wait$ and $MaxShift$ of $k$.

$$Shift_j = c_{ij} + c_{jk} - c_{ik} + Wait_j + T_j$$ (13)

For each not yet included location the total time consumption is calculated for each possible position in the current schedule, of which the lowest one is recorded. Then, a ratio is calculated by dividing the squared score of a location by the minimum time consumption, to determine which location to insert into the schedule. The location with the highest ratio is selected.

$$Ratio_i = \frac{Score_i^2}{Shift_i}$$ (14)

Because of the time windows, the score of a location is more important than the amount of shift an insertion causes and therefore the score of the location is squared. When the square of the score is not used, the results become worse, according to Vansteenwijgen et al.

When the best possible insertion is chosen and inserted, the variables of the visits after the insertion should be updated. The variables that should be updated are the arrival time $a$, starting time $s$, $Wait$, $MaxShift$ and $Shift$. Afterwards, the $MaxShift$ for the inserted location should be updated and lastly, the $MaxShift$s for the visits before the insertion should be updated. Algorithm 1 shows the pseudo code for the insertion step.

Algorithm 1 Insertion step

1. for each not yet included location do
2. Determine best possible position to insert and calculate $Shift$
3. Calculate $Ratio$
4. Insert the location with the highest ratio into the schedule
5. Calculate arrival $a$, start of visit $s$ and $Wait$ for the inserted location
6. for each location after the insertion (until $Shift$ equals 0) do
7. Update arrival $a$, start of visit $s$, $Wait$, $MaxShift$ and $Shift$
8. Update $MaxShift$ of the inserted location
9. for each location before the insertion do
10. Update $MaxShift$

The following formulas are used to update the visits after the insertion. With regard for notation: a visit to location $j$ is inserted in between visits to locations $i$ and $k$. The shift of visit $j$ is calculated in formula (15), which is identical to formula (13) mentioned earlier. With this shift, the new waiting time ($Wait_{k^*}$) and new arrival time ($a_{k^*}$) of visit $k$ can be calculated in formulas (16) and (17). The shift of visit $k$ is calculated in formula (18), with
which the updated starting time \( (s_{k^*}) \) and updated \( MaxShift_{k^*} \) can be calculated in formulas (19) and (20). The formulas (16)-(20) are also used to update the variables of any visit after \( k \). Note that it may be possible that the insertion of visit \( j \) has no influence on the variables of visit \( k \) and any later visit at all.

There is a small difference in the meaning of \( Shift_j \) and \( Shift_k \). Where \( Shift_j \) represents the amount of time a possible insertion of location \( j \) costs, \( Shift_k \) equals the amount of time the starting time of the visit of location \( k \) is actually shifted because of the insertion of \( j \).

\[
Shift_j = c_{ij} + c_{jk} - c_{ik} + Wait_j + T_j \tag{15}
\]

\[
Wait_{k^*} = \max[0, Wait_k - Shift_j] \tag{16}
\]

\[
a_{k^*} = a_k + Shift_j \tag{17}
\]

\[
Shift_k = \max[0, Shift_j - Wait_k] \tag{18}
\]

\[
s_{k^*} = s_k + Shift_k \tag{19}
\]

\[
MaxShift_{k^*} = MaxShift_k - Shift_k \tag{20}
\]

### 4.2 Shake step

It is possible that the algorithm ends up in a local optimum after the insertion step. To escape such an optimum, a shake step is created, which enables the solution climb out of the local optimum and move towards a better solution.

In the shake step, one or more consecutive visits are deleted from the schedule. There are two parameters for the shake step, \( R \) and \( S \). \( R \) indicates the number of visits that are deleted in the schedule and \( S \) indicates at which visit the removal starts. When the end of the schedule is reached during the removal process and there are still visits to be removed, the shake step continues at the beginning of the schedule.

The visits after the deleted ones are shifted towards the beginning of the time frame as much as possible, with regard to the individual time windows. For the visits after the deleted ones the arrival time \( a \), starting time \( s \), \( Wait \), \( MaxShift \) and \( Shift \) should be updated, just as in the insertion step. For the visits before the deleted ones only \( MaxShift \) should be updated. The pseudo code for this step is shown in Algorithm 2.

#### Algorithm 2 Shake step

1. for each time frame do
2. Delete a set of \( R \) visits, starting at visit \( S \)
3. Calculate \( Shift \)
4. for each visit after the deleted visit(s) (until \( Shift \) equals 0) do
5. Shift visits towards the start of the time frame as much as possible
6. Update arrival \( a \), start of visit \( s \), \( Wait \), \( MaxShift \) and \( Shift \)
7. for each visit before the deleted visit(s) do
8. Update \( MaxShift \)
4.3 ILS heuristic

The heuristic starts with an empty schedule and initializes both parameters of the shake step to 1. The heuristic loops through the insertion step and shake step until no improvements have been made for a fixed number of times. At first, the insertion step is repeated until a local optimum is reached. If this solution is better than the best found solution so far, the new solution is saved and R and the number of times no improvement has been made are set to 1 and 0, respectively. Afterwards, the shake step is executed on the found solution after the insertion step, whether or not this was a better solution than the best found so far. Then, S and R are updated by adding the value of R to S and adding 1 to R. If S is larger than or equal to the number of visits scheduled in the time frame with the least amount of visits, S is deducted by this number. This number is later referred to as the smallest tour size of a schedule. When R equals \( \frac{n}{3m} \), it is set to 1. The pseudo code of the ILS heuristic is shown in Algorithm 3.

Algorithm 3 ILS heuristic

1: Set \( S = 1 \), \( R = 1 \) and \( N_{\text{NoImprovement}} = 0 \)
2: while \( N_{\text{NoImprovement}} < 150 \) do
3:     while no local optimum found do
4:         Execute insertion step
5:         if current solution better than best found so far then
6:             Save current solution to best found so far and set \( R = 1 \) and \( N_{\text{NoImprovement}} = 0 \)
7:         else
8:             \( N_{\text{NoImprovement}} = N_{\text{NoImprovement}} + 1 \)
9:         end if
10:     Execute shake step with parameters \( R \) and \( S \)
11:     Set \( S = S + R \) and \( R = R + 1 \)
12:     if \( S \geq \text{smallest tour size} \) then
13:         Set \( S = S - \text{smallest tour size} \)
14:     end if
15:     if \( R = \frac{n}{3m} \) then
16:         Set \( R = 1 \)
17: end if
18: return the best solution found

By updating the shake step parameters as described above, different visits are removed every time the shake step is executed. It is very probable that during the execution of the entire heuristic each visit is deleted at least once. This technique addresses the problems that arise when simple techniques are used, as described by Gendreau et al. [3]. The solution space is now better explored and wrong decisions made in the insertion step can be corrected. This is also caused by the fact that the heuristic continues to optimize the current solution found by the insertion step instead of the best found solution so far.

The only parameters that need to be decided beforehand are the maximum number of iterations without improvement and the maximum number of visits to delete in the shake step. Vansteenwegen et al. claim that the used value (150) and formula \( \left( \frac{n}{3m} \right) \) in the pseudo code above for these parameters produce the best results.

4.4 Variants of the heuristic

In their paper, Vansteenwegen et al. make claims about some aspects of their heuristic. These claims will be checked but we also look for possible improvements of these aspects in order to improve the results of the algorithm. The aspects checked are the heuristics used to reset the parameters \( R \) and \( S \), the formula to calculate the ratio in the insertion step and lastly, the value of the maximum number of iterations without improvement.
4.4.1 Resetting the parameter R

The maximum number of visits deleted in the shake step is restricted by Vansteenwegen et al. to \( \frac{n}{3m} \), the number of locations divided by triple the number of time frames planned. They argue that it made no difference in their research whether they used \( \frac{n}{m} \) or \( \frac{n}{3m} \). However, they do not clearly explain why they used \( \frac{n}{3m} \) instead. In this thesis we try to verify their claim and consider other possible formulas to decide the maximum number of visits to delete.

Firstly, we notice that the ILS heuristic sets R back to 1, once R equals \( \frac{n}{3m} \), while this fraction is hardly ever an integer as most test instances used consist of 100 locations. Therefore, we adjust this to setting R back to 1, once R is greater than or equal to the fraction.

Secondly, we check whether the fractions \( \frac{n}{m} \), \( \frac{n}{3m} \) and \( \frac{n}{5m} \) cause different results for the algorithm.

So far, the fraction has been dependent on the value of \( m \), meaning that the maximum number of visits deleted is dependent on how many time frames are being planned simultaneously. Besides a time-wise argument, we see no reason why fewer visits should be deleted in the shake step when more time frames are planned simultaneously. Using a test instance with 100 locations, the shake step deletes up to 8 visits when planning four time frames, but it deletes up to 33 visits when planning a single time frame. Time-wise, this method can be argued, as deleting more visits over multiple time frames means more visits have to be rescheduled in the insertion step. However, for finding an algorithm that finds the best possible score, as is the focus in this thesis, this method is not intuitive.

We try two different approaches for a fraction that is independent of \( m \). The first is a formula that is also independent of the number of scheduled visits, as the initial formula does not take this into account either. This formula has the form of \( \frac{n}{\text{constant}} \). If the constant is too small, and many visits can be deleted, the insertion step is more likely to start over rescheduling the entire time frame, instead of only rescheduling part of it. Therefore, it is less likely to find a different schedule and to improve the best found solution so far.

The second approach takes the number of scheduled visits into account. The time windows within the test instances differ quite a lot, which is discussed more elaborate in Chapter 5, which causes that the number of visits in a single time frame within a solution varies between 6 and 57. This suggests that using a constant as the maximum number of visits to be deleted might not work optimally. If the maximum number of visits to delete equals 20, the entire schedule is deleted for some test instances, while for other instances, only one-third is deleted. Therefore, we delete a percentage of the number of scheduled visits within the shake step.

4.4.2 Resetting the parameter S

The exact way of updating the parameter S in the paper of Vansteenwegen et al. is not clear to us. Firstly, it is not clear whether they decided to use the parameter S as a scalar or as a vector. Secondly, Vansteenwegen et al. describe the parameter S to always have a value within the length of the shortest tour. However, when applying the method described in their paper, the value of S can become much higher than the size of the shortest tour, due to the possible high values of R. Furthermore, as the size of the smallest tour is decided after the shake step (in which visits are deleted), the smallest tour size is not representative and can even equal 0 if the value of R becomes too high. This causes the shake step to start deleting at the beginning of the schedule too often. Due to this description
being unclear in the original paper, the reproduction of the algorithm might not work properly, because the value of $S$ does not iterate in the desired way.

We implement the heuristic initially with parameter $S$ as a scalar. At first, we try a variant where the size of the smallest tour is decided before the shake step. Secondly, we try a variant where the value of $S$ is a random integer in the interval from 0 to the smallest tour size. Thirdly, a variant is tried where $S$ is used as a vector, since Vansteenwegen et al. suggest that different values within $S$ for different time frames increase the possibility of escaping local optima. The vector will first be used in the same way as the scalar $S$ is used in the reproduction algorithm, although the smallest tour size will now be replaced by the tour size of the corresponding time frame. If this is not adjusted, all values within the vector will stay equal to each other and the effect of the method is lost. Afterwards, the two variants suggested before for the scalar parameter $S$ will be implemented in the vector parameter as well.

4.4.3 Ratio formula

In the insertion step the squared score of a location is used in order to calculate the ratio of a location. Vansteenwegen et al. argue that due to the time windows, the time consumption of a potential inserted location is of less importance than its score. Therefore, the used formula should work better than, for example, $\frac{\text{Score}}{\text{Shift}}$. We run the algorithm with multiple different formulas in order to find the best formula to calculate the ratio.

4.4.4 Maximum number of iterations without improvement

The ILS heuristic of Vansteenwegen et al. runs for a maximum of 150 iterations without improvement. They argue that a higher maximum number of iterations has a positive effect on the outcome of the algorithm, but also increases the computation time significantly. A lower number of iterations should have the opposite effect. We investigate the effect of increasing and decreasing the number of iterations on the total score and the computation time.
5 Data

The datasets used in this research are found at http://www.mech.kuleuven.be/en/cib/op. The datasets C1 (9), C2 (8), R1 (12), R2 (11), RC1 (8) and RC2 (8) are used, which includes 56 test instances in total, where the numbers between brackets indicate the division of the test instances over the different sets. These sets are based on Solomon’s datasets [11] for the vehicle routing problem with time windows. Each dataset is used for planning schedules consisting of $m$ time frames, where $m$ varies between 1 and 4. So there are a total of 224 different problems to be solved.

All test instances have 101 locations. The first is the departure/arrival location. The other 100 locations can be visited within the time frames. For each location the x- and y-coordinates are given, together with the visiting time, the score that is collected when it is visited and the time window during which a visit can start.

5.1 Different datasets

There are different types of sets created to highlight some factors that might affect the solution of the routing and scheduling algorithms. The different factors include the geographical data and the number of locations visited in a single time frame.

In R-datasets, the locations are randomly generated. In C-sets the locations are clustered and in RC-sets there is a mix of clustered and randomly positioned ones.

The 1-sets have a short scheduling horizon, which allow only a few visits per time frame (on average 10) to be scheduled. 2-sets have a large planning horizon, allowing more than 30 visits per time frame.

Lastly, within the R-, C- and RC-set, the x- and y-coordinates of the locations in the 1-set should be exactly equal to the ones in the 2-set and only the width of the time windows should differ. We verified this for the R- and RC-sets, but within the provided C1- and C2-sets only 72 locations have identical x- and y-coordinates.

After running the various variants of the algorithm, it will be checked whether a specific variant might work better for a certain dataset, due to its characteristics, while another variant is better for the remainder of the datasets.

5.2 Time windows

The width of the time windows differs quite a lot throughout the datasets. There are test instances where locations are opened in an interval which covers only 5% of the time frame. However, there are also instances where the time windows of locations cover more than 70% of the time frame. In the latter, there are far more opportunities to schedule the visits, as the time of the visit is hardly restricted. Short opening times can have an influence on the number of scheduled visits too, as visiting a certain location might exclude the opportunity to visit other locations because the time windows overlap or follow each other too shortly, as there are travel times to be considered too.
Furthermore, the length of a time frame varies throughout the datasets. In some datasets the time frames have a length of 230 time units, while others have a length of 3390 time units. This does not necessarily have an impact on the amount of locations visited in a solution if the visiting times are adjusted accordingly. However, the length of a visit as a percentage of the time frame is not the same throughout the different datasets, which causes that in some datasets at most 13 locations can be visited (even if travel times and time windows are left out), while in others all 100 locations could be visited.

5.3 Optimal solutions

Vansteenwegen et al. use the best-known results until 2009 for each test instance to be able to review their results. These best-known results are either the optimal solutions or the best results from five runs of the ant colony system (ACS) [10]. Mostly, the outcome of the ACS is used, as the optimal solution is only known for 68 of the 224 problems. Some of the other results might be optimal too, but it has not been proven yet.

For $m = 1$, 29 of the 56 test instances have a known optimal solution, which happen to be all the instances of the 1-sets. The ILS heuristic of Vansteenwegen et al. found 10 of these. No optimal solutions are known for $m = 2$ and there are 12 known for $m = 3$, distributed over instances within the different 2-sets, of which the ILS heuristic found 11. For $m = 4$ there are 27 optimal solutions known, which happen to be all the instances of the 2-sets. The ILS heuristic found all of them.
6 Results

We run the algorithm on a HP Probook 6560b laptop (2.5 GHz processor and 4.0 GB RAM) using Matlab, version 7.12.0.635 (R2011a). This is a different computer and program than the one used by Vansteenwegen et al. and we have therefore deviating computation times. The computation times are mentioned in the results in order to compare the different variants to each other.

We compared the results of our reproduced algorithm to the results from the ILS heuristic of Vansteenwegen et al. and not to the best-known results printed in their paper. Later, when we are evaluating the different variants, we will compare the results to our reproduced algorithm to be able to make fair conclusions. At the end, we will compare the results of the best variant (which may be a combination of variants) with the ILS heuristic and the best-known results until 2009.

6.1 Reproduction of the ILS heuristic

The results after implementing the heuristic are sadly enough not exactly equal to the results of Vansteenwegen et al. The performance of the reproduction compared to the ILS heuristic are shown in Table 1. Only 65 of the test instances (29.0%) received the same results and most instances received worse results.

<table>
<thead>
<tr>
<th></th>
<th>better</th>
<th>equal</th>
<th>worse</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>15</td>
<td>65</td>
<td>144</td>
<td>224</td>
</tr>
<tr>
<td>%</td>
<td>6.7%</td>
<td>29.0%</td>
<td>64.3%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 1: Performance of the reproduction algorithm compared to the ILS heuristic.

It is not completely clear what causes the difference between their results and ours. We programmed the insertion step and shake step according to their paper, but there could be an implementation or an interpretation error, as the method to reset parameter $S$ is not completely clear, as previously discussed in Section 4.4.2.

Table 2 shows the average percentage difference between the results of the reproduction and the ILS heuristic, per dataset and amount of time frames. A positive percentage means that the reproduction algorithm produced a better result in the corresponding dataset, which is only the case for three subparts of sets R2 and RC2. A 0 indicates that the score of the dataset was neither improved or worsened. It is, however, possible that in one of the test instances the score increased and in another one the score decreased with the same value.

<table>
<thead>
<tr>
<th></th>
<th>$m = 1$</th>
<th>$m = 2$</th>
<th>$m = 3$</th>
<th>$m = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>-1.53</td>
<td>-1.67</td>
<td>-2.23</td>
<td>-0.42</td>
</tr>
<tr>
<td>C2</td>
<td>-1.23</td>
<td>-0.17</td>
<td>-0.56</td>
<td>0</td>
</tr>
<tr>
<td>R1</td>
<td>-2.14</td>
<td>-2.52</td>
<td>-6.21</td>
<td>-5.34</td>
</tr>
<tr>
<td>R2</td>
<td>-1.76</td>
<td>0.32</td>
<td>0.03</td>
<td>0</td>
</tr>
<tr>
<td>RC1</td>
<td>-5.58</td>
<td>-4.42</td>
<td>-3.54</td>
<td>-5.61</td>
</tr>
<tr>
<td>RC2</td>
<td>-3.16</td>
<td>-1.19</td>
<td>0.21</td>
<td>-0.01</td>
</tr>
<tr>
<td>total</td>
<td>-2.20</td>
<td>-0.99</td>
<td>-1.45</td>
<td>-1.68</td>
</tr>
</tbody>
</table>

Table 2: Percentage difference between the reproduction and the ILS heuristic, per dataset and amount of time frames.
We can clearly see that the 2-sets perform better in the reproduction than the 1-sets. The percentages for a certain dataset and certain amount of time frames are always higher for the 2-sets than the 1-sets. The 2-sets have a larger planning horizon, as mentioned in Chapter 5, which might make it easier for the algorithm to fit the visits into the schedule.

When looking at the datasets and amount of time frames separately, large differences between the cells are noticed. It is unclear why the algorithm performs well for a certain dataset, for example R2, when planning multiple time frames, but performs worse when planning only a single time frame. Simultaneously, set R1 performs best when planning only a single time frame and worse when planning multiple time frames.

In the column of \( m = 4 \) two 0’s and a -0.01 are found in the cells corresponding to the 2-sets, indicating that the reproduction found almost exactly the same results as the ILS heuristic. As mentioned in Section 5.3, all 27 instances of the 2-sets have an optimal solution available and these are all found by the ILS heuristic. Our reproduction found 26, which leads to these percentages as only one test instance scored worse than the ILS. For \( m = 3 \) both the ILS heuristic and the reproduction found 11 of the 12 optimal solutions. This causes that only for the remaining 16 instances within the 2-sets scores could deviate, leading to percentages close to 0 in the corresponding cells. For \( m = 1 \) only 5 of the 29 optimal solutions for the 1-sets are found, while the ILS heuristic found 10. Clearly, both the ILS heuristic as our reproduction do not perform very well compared to the optimal solutions. The reproduction, however, does also not perform well compared to the ILS heuristic as the percentages are all very low.

The computation time of our reproduction for all 224 problems together was 5800 seconds. This is way longer than the computation time of Vansteenwegen et al., which was 335 seconds, but this is most likely caused by the different computer and program used by us.

In the following sections, where the different variants are evaluated, the results are compared to the results of the reproduction. It should be noted that for the 42 instances where optimal solutions are found, no improvements can be made by the variants; the scores can only worsen or stay the same. However, as further on the results are evaluated with all the different amount of time frames combined, this effect is not clearly visible in the tables.

6.2 Resetting the parameter R

6.2.1 Formulas dependent on \( m \)

First, we compared the formulas \( \frac{2}{m} \) and \( \frac{5}{2m} \) for the maximum amount of visits to be deleted in the shake step to the original formula \( \frac{n}{3m} \). Additionally, other variants of this formula have been tried too.

Table 3 shows the performance of the variants compared to the reproduction with formula \( \frac{n}{3m} \). It can be seen that the formula \( \frac{2}{m} \) performs very poorly. Compared to the formula of \( \frac{n}{3m} \), 27 test instances receive a worse result, while only one of the 224 solutions improves. The higher the number in the denominator, the more instances are improved. These findings tell us that there is certainly a difference between the formulas and therefore we cannot support the claim of Vansteenwegen et al. that it makes no difference which one of the three formulas is used.
Table 3: Performance of the different variants dependent on the number of time frames, compared to the reproduction with formula $\frac{n}{3m}$.

Table 4 shows the results of the different formulas per dataset compared to the reproduction. For the 1-sets the formula $\frac{n}{m}$ is the best variant, but for the 2-sets it differs. However, the results show us that any formula with a denominator higher than $3m$ improves the results, while a lower denominator only worsens them. Lastly, the 1-sets have a higher percentage improvement than the 2-sets, but this might be caused by the smaller planning horizons. Since smaller planning horizons cause less locations to be visited, the total scores of the datasets are smaller, such that a small improvement in a 1-set results in a higher percentage improvement than in a 2-set.

Table 4: Detailed results per dataset for all the variants dependent on the number of time frames, in percentages.

Table 4 also shows the computation times for the different variants. As we expected, the higher the number in the denominator, the quicker the algorithm is run, because when fewer visits are deleted in the shake step, fewer visits have to be rescheduled in the insertion step.

We also looked for the best formula per amount of time frames planned, which results can be found in Table 5. However, we do not see much logic in the outcome. While $\frac{n}{m}$ is one of the best variants for $m = 4$, which means deleting up to 5 visits in the shake step, variant $\frac{n}{3m}$ is the best for $m = 1$, which means deleting up to 25 visits in the shake step. We would expect that a variant with a higher denominator, such as $\frac{n}{8m}$ (deleting at most 12 visits), would have worked better for $m = 1$, such that the maximum number of visits to be deleted are more similar over the different amounts of time frames planned.

Table 5: Summary of results per amount of time frames.

6.2.2 Formulas independent of $m$ and the number of scheduled visits

Secondly, we tested formulas independent of both $m$ and the number of scheduled visits. The results of these variants are presented in Table 6. The formulas with a high denominator generate positive results for any type of set, with the exception of formula $\frac{n}{25}$ for set RC1. Formulas $\frac{n}{5}$ and $\frac{n}{10}$ only have an influence on the results of the 2-sets, except for formula $\frac{n}{10}$ in combination with set R1. This can be explained by the short planning horizon of the 1-sets and the number of locations within a solution. As each set has 100 locations that can be visited, formulas $\frac{n}{5}$ and $\frac{n}{10}$
will result in the deletion of maximum 20 and 10 visits within the shake step, respectively. As the solutions in the
1-sets have an average of 10 visits per time frame, an entire schedule can be deleted in the shake step. This reduces
the effectiveness of this step significantly, as can be seen in the results.

<table>
<thead>
<tr>
<th></th>
<th>n=5</th>
<th>n=10</th>
<th>n=15</th>
<th>n=20</th>
<th>n=25</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0.0</td>
<td>0.01</td>
<td>1.46</td>
<td>1.54</td>
<td>1.20</td>
</tr>
<tr>
<td>C2</td>
<td>-0.47</td>
<td>0.76</td>
<td>0.49</td>
<td>0.87</td>
<td>0.72</td>
</tr>
<tr>
<td>R1</td>
<td>0.0</td>
<td>0.01</td>
<td>2.70</td>
<td>3.13</td>
<td>1.10</td>
</tr>
<tr>
<td>R2</td>
<td>0.13</td>
<td>0.64</td>
<td>0.41</td>
<td>0.40</td>
<td>0.48</td>
</tr>
<tr>
<td>RC1</td>
<td>0.0</td>
<td>0.0</td>
<td>1.60</td>
<td>1.72</td>
<td>-0.04</td>
</tr>
<tr>
<td>RC2</td>
<td>0.10</td>
<td>0.79</td>
<td>0.67</td>
<td>0.57</td>
<td>0.61</td>
</tr>
<tr>
<td>total</td>
<td>-0.04</td>
<td>0.49</td>
<td>0.98</td>
<td>1.11</td>
<td>0.68</td>
</tr>
<tr>
<td>CPU (s)</td>
<td>9603</td>
<td>5017</td>
<td>4427</td>
<td>3391</td>
<td>2931</td>
</tr>
</tbody>
</table>

Table 6: Detailed results per dataset for all the variants independent of the number of time frames and the number
of scheduled visits, in percentages.

Overall, formula \( \frac{n}{20} \) turns out to work best, which deletes up to 5 visits per schedule. This is equivalent of
deleting up to 50% of the visits within the 1-sets and up to 12.5% within the 2-sets. This variant, however, is not
the best for two of the 2-sets. In these sets, the variant \( \frac{n}{10} \) works better, which means deleting up to 10 visits (or
25%). This suggest that maybe the percentage of deleted visits is of importance, as deleting up to 25% works better
than deleting up to 12.5%. However, deleting up to 50% (formula \( \frac{n}{5} \)), just as \( \frac{n}{20} \) does for the 1-sets, scores not that
good. Furthermore, the variant \( \frac{n}{25} \) works quite well too for the 2-sets while maximum 10% of the visits are deleted.

The bottom row in Table 6 confirms, just like Table 4 in the previous section, that the fewer visits are deleted
in the shake step, the quicker the algorithm runs.

6.2.3 Formulas dependent on the number of scheduled visits

Lastly, we tested formulas that are dependent on the number of scheduled visits in the insertion step. Table 7 shows
the performance of these variants compared to the reproduction. If we compare Table 7 to Table 3 in Section 6.2.1,
it can be seen that, even though most variants have a higher number of improved test instances, these variants do
not necessarily perform better, because all variants receive more worse results too.

<table>
<thead>
<tr>
<th></th>
<th>10%</th>
<th>15%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>worse</td>
<td>50</td>
<td>52</td>
<td>33</td>
<td>21</td>
<td>19</td>
</tr>
<tr>
<td>better</td>
<td>85</td>
<td>85</td>
<td>101</td>
<td>94</td>
<td>53</td>
</tr>
</tbody>
</table>

Table 7: Summary of results of all the variants dependent on the number of scheduled visits.

Table 8 shows the more detailed results of these variants per dataset. We see that the 50% variant gives the
best percentage improvement. However, this variant works particularly well for the 1-sets and not so good for the
2-sets. For the 2-sets the variant with 15% seems to be better, even though this does not result in the best score for
the set R2.

Once again, it can be seen that the algorithm runs quicker when less visits are deleted in the shake step.
Table 8: Results per dataset for all the variants dependent on the scheduled visits, in percentages.

<table>
<thead>
<tr>
<th></th>
<th>10%</th>
<th>15%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0.41</td>
<td>0.41</td>
<td>1.87</td>
<td>1.50</td>
<td>0.56</td>
</tr>
<tr>
<td>C2</td>
<td>0.66</td>
<td>0.87</td>
<td>0.53</td>
<td>0.32</td>
<td>-0.11</td>
</tr>
<tr>
<td>R1</td>
<td>-1.50</td>
<td>-1.50</td>
<td>1.17</td>
<td>2.82</td>
<td>0.76</td>
</tr>
<tr>
<td>R2</td>
<td>0.55</td>
<td>0.42</td>
<td>0.82</td>
<td>0.37</td>
<td>1.43</td>
</tr>
<tr>
<td>RC1</td>
<td>-0.52</td>
<td>-0.52</td>
<td>-0.02</td>
<td>1.31</td>
<td>0.98</td>
</tr>
<tr>
<td>RC2</td>
<td>0.76</td>
<td>1.25</td>
<td>0.50</td>
<td>0.62</td>
<td>0.36</td>
</tr>
<tr>
<td>total</td>
<td>0.27</td>
<td>0.38</td>
<td>0.79</td>
<td>0.92</td>
<td>0.66</td>
</tr>
<tr>
<td>CPU (s)</td>
<td>2002</td>
<td>2392</td>
<td>3560</td>
<td>7463</td>
<td>7114</td>
</tr>
</tbody>
</table>

Table 9: Results per dataset for all the variants of the scalar $S$, in percentages.

<table>
<thead>
<tr>
<th></th>
<th>s-&quot;better S&quot;</th>
<th>s-random S</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0.94</td>
<td>1.27</td>
</tr>
<tr>
<td>C2</td>
<td>0.13</td>
<td>-0.21</td>
</tr>
<tr>
<td>R1</td>
<td>1.99</td>
<td>3.29</td>
</tr>
<tr>
<td>R2</td>
<td>-0.05</td>
<td>-0.31</td>
</tr>
<tr>
<td>RC1</td>
<td>1.36</td>
<td>3.11</td>
</tr>
<tr>
<td>RC2</td>
<td>-0.02</td>
<td>-0.13</td>
</tr>
<tr>
<td>total</td>
<td>0.47</td>
<td>0.65</td>
</tr>
<tr>
<td>CPU (s)</td>
<td>6688</td>
<td>9399</td>
</tr>
</tbody>
</table>

Table 9: Results per dataset for all the variants of the scalar $S$, in percentages.

6.3 Resetting the parameter $S$

We start with the two variants where parameter $S$ is used as a scalar; one where the smallest tour size is decided before the shake step (variant “better $S$”) and one where the value of $S$ is determined randomly. The results are shown in Table 9. The variant with random values of $S$ performs quite well in the 1-sets. It is unclear though why it performs so poorly in the sets with a larger planning horizon. In the variant “better $S$” the smallest tour size is decided before the shake step, and thus is hardly ever zero as no visits have been deleted yet. It was expected that this variant would improve the scores a lot as the shake step starts deleting visits in the middle of the schedule more often, instead of starting at the beginning of the schedule every time. However, we see from the table that this variant has a weaker influence on the scores than the random variant. The only deviation is set C2, where this variant has an opposite effect compared to the random variant.

Both variants have an increased computation time compared to the reproduction, but it is not clear to us why especially the random variant took so long to run.

Afterwards we implemented parameter $S$ as a vector instead of a scalar. We tried three variants: the first where $S$ was updated as in the reproduction algorithm, the second where the number of visits in each time frame is determined before the shake step, as in the scalar variant “better $S$”, and the third where the values of $S$ are determined randomly. The detailed results are found in Table 10. These variants do not perform very well and even worse compared to the total improvement of the variants of the scalar parameter $S$ in Table 9. Vansteenwegen et al. claim that different values within the vector $S$ would lead to better results. Based on these results, we can not reject this claim, as the variants of the vector $S$ do perform better than the reproduction algorithm where a scalar was used. However, the variants of the scalar $S$ perform even better. For the 2-sets, the variant with the initial way to reset $S$ performs best, although the scores are quite low and for set RC2 even negative. The other two variants...
perform better for the 1-sets, but have a negative influence on the scores of the 2-sets.

<table>
<thead>
<tr>
<th></th>
<th>v-initial S</th>
<th>v-“better S”</th>
<th>v-random S</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>-0.19</td>
<td>0.71</td>
<td>0.82</td>
</tr>
<tr>
<td>C2</td>
<td>0.15</td>
<td>-0.47</td>
<td>-0.95</td>
</tr>
<tr>
<td>R1</td>
<td>0.49</td>
<td>2.82</td>
<td>2.20</td>
</tr>
<tr>
<td>R2</td>
<td>0.03</td>
<td>-0.14</td>
<td>-0.04</td>
</tr>
<tr>
<td>RC1</td>
<td>-0.05</td>
<td>2.21</td>
<td>1.94</td>
</tr>
<tr>
<td>RC2</td>
<td>-0.03</td>
<td>-0.41</td>
<td>-0.72</td>
</tr>
<tr>
<td>total</td>
<td>0.07</td>
<td>0.38</td>
<td>0.16</td>
</tr>
<tr>
<td>CPU (s)</td>
<td>5744</td>
<td>7220</td>
<td>7642</td>
</tr>
</tbody>
</table>

Table 10: Results per dataset for all the variants of the vector S, in percentages.

The computation time of the variant that updates the vector S in the initial way is as long as the time of the reproduction. The other two variants have a longer computation time, but are very different from the computation times of the scalar variants. It would have been understandable if the vector variants were slower, as more information has to be stored and updated separately, but this is only the case for the variant “better S”. For the variant with random values within S, the vector variant is much quicker, which is strange as more random values have to be determined.

### 6.4 Ratio formula

Besides calculating the ratios in the insertion step with the alternative formula of \( \frac{\text{Score}}{\text{Shift}} \), as suggested by Vansteenwegen et al., we also check some other formulas. The results are shown in Table 11. As can clearly be seen, the different formulas generate more worse solution than better ones. The total score of the solutions became worse for each variant too. We can therefore agree with the statement of Vansteenwegen et al. that due to the time windows, the score of a visit is more important than the time consumption its insertion costs. However, there should not be put too much weight on the score alone, as the formula of \( \frac{\text{Score}^3}{\text{Shift}} \) performs worse too.

<table>
<thead>
<tr>
<th></th>
<th>( \frac{\text{Score}}{\text{Shift}} )</th>
<th>( \frac{\text{Score}^2}{\text{Shift}} )</th>
<th>( \frac{\text{Score}^3}{\text{Shift}} )</th>
<th>( \frac{\text{Score}^3}{\text{Shift}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>worse</td>
<td>114</td>
<td>89</td>
<td>89</td>
<td>86</td>
</tr>
<tr>
<td>better</td>
<td>53</td>
<td>78</td>
<td>78</td>
<td>57</td>
</tr>
<tr>
<td>improv.</td>
<td>-1.23%</td>
<td>-0.23%</td>
<td>-0.23%</td>
<td>-0.42%</td>
</tr>
<tr>
<td>CPU (s)</td>
<td>5714</td>
<td>5792</td>
<td>5998</td>
<td>5427</td>
</tr>
</tbody>
</table>

Table 11: Summary of the results per variant of the ratio formula, compared to the initial formula \( \frac{\text{Score}^3}{\text{Shift}} \).

The computation times of the variants are comparable to the computation time of the reproduction, as there is no difference in the amount of variables to be calculated. The only difference is the method to calculate a single variable and we expected it to not have much influence on the computation time.

### 6.5 Maximum number of iterations without improvement

We have executed the algorithm with a maximum of 100, 125, 175 and 200 iterations without improvement, in comparison to the original number of 150. Figure 1 shows a graph with the percentage difference in computation time and total score. It can be seen that a lower maximum number of iterations reduces the computation time a lot, with 27.7% and 10.0% for 100 and 125 iterations respectively, while the total score of the found solutions only
is 0.06% and 0.05% worse. A higher maximum number of iterations raises the computation time extremely, with 22.0% and 35.6% for 175 and 200 iterations respectively, while the total solution only improves with 0.09% for both variants. Considering that the purpose of this thesis is to find an algorithm to find the best possible score, but at the same time we do have some time constraints to finish this thesis, we will continue to work with the guidelines of the paper of Vansteenwegen et al. and do not change the maximum number of iterations in further variants.

Figure 1: Percentage difference in computation time and total score for varying amounts of maximum number of iterations without improvement.

6.6 Combining different variants

We see that adjusting the formula of the ratio does not have a positive effect on the results. Adjusting the ways of resetting parameters R and S do have a positive effect on the overall results, however, not always for each dataset separately.

We are surprised to see that the different variants do not have any notable effect on the different C-, R-, and RC-sets. These sets differ in geographical locations, as discussed in Section 5.1, and we expected some variants to perform better for a certain type of test instances. However, all notable effects which could be explained, were due to the different planning horizons in the 1- and 2-sets. Therefore, we separate the 1-sets from the 2-sets and run combinations of the variants tried before, in order to investigate whether or not a combination of two variants works even better.

The 1-sets are run with combinations of the variants $\frac{R}{S}$, $\frac{R}{S}$ and 50% for the formula of R and scalar variant random S and the vector variant “better S”. These variants showed to be the best working variants in comparison to the other variants of the same kind. Even though the results from the variants separately indicated that the scalar variant random S should work better than the vector variant “better S”, we run all combinations as we do not know how the variants perform when they are combined with the variants of R.
The results of these combinations are shown in Table 12. The combination of deleting up to 50% of the visits in the shake step together with the random values of scalar S performs best for the 1-sets. This combination works even better than each variant does separately.

<table>
<thead>
<tr>
<th></th>
<th>s-random S</th>
<th>v-“better S”</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.51</td>
<td>3.95</td>
</tr>
<tr>
<td></td>
<td>3.84</td>
<td>3.04</td>
</tr>
<tr>
<td></td>
<td>2.78</td>
<td>2.92</td>
</tr>
</tbody>
</table>

Table 12: Results of the combined variants for the 1-sets.

The 2-sets are run with combinations of the variants \(\frac{n}{m}\), \(\frac{n}{m}\) and 15% for the formula of R and scalar variant “better S” and the vector variant initial S. These variants showed to be the best working variants in comparison to the other variants of the same kind. And once again, as we do not know how the variants perform when they are combined, we run all six combinations, even though the variants of S performed very poorly in comparison to the variants of R.

The results of these combinations are shown in Table 13. We see that the combination of deleting up to 15% of the visits together with the scalar variant “better S” is the best for the 2-sets. This combination works also better than each variant separately.

<table>
<thead>
<tr>
<th></th>
<th>s-“better S”</th>
<th>v-initial S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.14</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>0.48</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>0.69</td>
<td>0.59</td>
</tr>
</tbody>
</table>

Table 13: Results of the combined variants for the 2-sets.

In this section the computation times are not shown as we have run the 1-sets and 2-sets separately and the computation times are therefore not comparable to the other computation times.

6.7 Results of the best combination

When the best combination for the 1-sets is combined with the best combination for the 2-sets, we obtain results that are improved a lot compared to our reproduction and the ILS heuristic of Vansteenwegen et al. Recall that in our original reproduction, only 6.7% of the test instances gave better results and 64.3% of the test instances actually gave worse results compared to the ILS heuristic. Now, 33.5% of the test instances give better results and only 26.3% of the instances give worse results compared to the ILS heuristic. An overview of these results can be seen in Table 14.

<table>
<thead>
<tr>
<th></th>
<th>better</th>
<th>equal</th>
<th>worse</th>
</tr>
</thead>
<tbody>
<tr>
<td>reproduction</td>
<td>6.7%</td>
<td>29.0%</td>
<td>64.3%</td>
</tr>
<tr>
<td>best variant</td>
<td>33.5%</td>
<td>40.2%</td>
<td>26.3%</td>
</tr>
</tbody>
</table>

Table 14: Results of the reproduction and the best combination, compared to the ILS heuristic.

Furthermore, the score of 149 test instances within the best combination improved compared to the reproduction, while for only 13 test instances the results became worse. For these instances, the average worsening was 0.93%, but this worsening is outshined by the large number of instances that are actually improved with an average improvement of 3.7%.
Table 15 shows the detailed results of the best combination compared to the ILS heuristic. The results have improved greatly, especially if it is compared to Table 2 in Section 6.1. The results over all the datasets together for each amount of time frames are now all positive, while they were negative in Table 2. Furthermore, there are fewer negative percentages and the percentages that are negative are much higher than they were before. One strange observation is the cell corresponding to dataset R2 and $m = 3$. The score of these test instances had improved the result of the ILS heuristic after the execution of the reproduction with 0.03%. However, after running the best combination, the score of these test instances has worsened with 0.03%.

<table>
<thead>
<tr>
<th></th>
<th>m=1</th>
<th>m=2</th>
<th>m=3</th>
<th>m=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0.31</td>
<td>-0.17</td>
<td>0</td>
<td>-0.40</td>
</tr>
<tr>
<td>C2</td>
<td>-0.96</td>
<td>1.04</td>
<td>0.92</td>
<td>0</td>
</tr>
<tr>
<td>R1</td>
<td>0.18</td>
<td>0.87</td>
<td>-0.41</td>
<td>-0.16</td>
</tr>
<tr>
<td>R2</td>
<td>1.19</td>
<td>1.27</td>
<td>-0.03</td>
<td>0</td>
</tr>
<tr>
<td>RC1</td>
<td>-0.92</td>
<td>-0.75</td>
<td>-0.04</td>
<td>1.94</td>
</tr>
<tr>
<td>RC2</td>
<td>-0.39</td>
<td>0.43</td>
<td>0.46</td>
<td>0</td>
</tr>
<tr>
<td>total</td>
<td>0.07</td>
<td>0.69</td>
<td>0.23</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Table 15: Percentage difference between the best combination and the ILS heuristic, per dataset and value of $m$.

The total score of the best combination improved 1.83% compared to our reproduction. Also, it found more optimal solutions than the reproduction. Where our reproduction found only 5 of the 29 optimal solutions for $m = 1$ and Vansteenwegen et al. 10, the best combination found 11. It also found all 27 optimal solutions for $m = 4$, just like the ILS heuristic, where the reproduction found 26. For $m = 3$ it found the same optimal solutions as the ILS heuristic and the reproduction did.

Compared to the results of Vansteenwegen et al. the improvement was only 0.28%. This reduced improvement is explained by the bad performance of our reproduction against the results of Vansteenwegen et al. Furthermore, it is hard to improve the ILS heuristic drastically as it is a good performing heuristic. The heuristic found many of the optimal solutions and it is possible that it found even more, but these other solutions have not yet been proven to be optimal. Despite the rather small percentage improvement compared to the ILS heuristic, the best combination did improve 32 of the best-known results until 2009.
The reproduction of the heuristic did not lead to exactly the same results as Vansteenwegen et al. The total score was actually 1.52% lower than their score. For 114 test instances (64.3%) the results were worse than the results of Vansteenwegen et al. For only 15 test instances (6.7%) our results were better.

Secondly, we studied the use of parameters \( R \) and \( S \), the ratio formula and the maximum number of iterations without improvement. We came to the following conclusions:

- It is better to use formula \( \frac{n}{m} \) to reset the parameter \( R \) rather than \( \frac{n}{3m} \), even though Vansteenwegen et al. claim that these formulas have no different influence on the results.
- Other ways of resetting the parameter \( R \), such as using the formula \( \frac{n}{20} \) or a percentage of scheduled visits, can improve the results even more.
- Any of the suggested variants for resetting the parameter \( S \) improve the result.
- All variants with an adjusted ratio formula within the insertion step worsen the result.
- Increasing or decreasing the maximum number of iterations without improvement does not have a large effect on the total score, but it does have great impact on the computation time.

Finally, a combination of variants was used over the different datasets in order to improve the algorithm as much as possible. For the 1-sets, the variant of deleting up to 50% of the scheduled visits in the shake step was used in combination with the variant assigning random values to the scalar parameter \( S \). For the 2-sets, the variant of deleting up to 15% of the scheduled visits in the shake step was used in combination with the scalar variant “better \( S \)”, in which the smallest tour size is decided before the shake step.

Compared to our reproduction, this combination is an enormous improvement; the scores of 149 test instances have improved, leading to an overall improvement of 1.83%. The performance compared to the ILS heuristic has improved greatly too. For 75 instances (33.5%) the score is improved compared to the results of Vansteenwegen et al., and only 59 (26.3%) instances received worse results. The total score of all test instances has improved with 0.28% compared to the results of Vansteenwegen et al. The best combination even improved the best-known results (until 2009) of 32 test instances.
References


