DEPARTMENT OF ENGINEERING MANAGEMENT

The multiple travelling salesperson problem with hotel selection

Marco Castro, Kenneth Sörensen, Peter Goos & Pieter Vansteenwegen
DEPARTMENT OF ENGINEERING MANAGEMENT

The multiple travelling salesperson problem with hotel selection

Marco Castro, Kenneth Sörensen, Peter Goos & Pieter Vansteenwegen

RESEARCH PAPER 2014-030
DECEMBER 2014

University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium
Research Administration – room B.226
phone: (32) 3 265 40 32
fax: (32) 3 265 47 99
e-mail: joeri.nys@uantwerpen.be

The research papers from the Faculty of Applied Economics
are also available at www.repec.org
(Research Papers in Economics - RePEc)

D/2014/1169/030
The multiple travelling salesperson problem with hotel selection

M. Castro¹, K. Sörensen¹, P. Goos¹,², and P. Vansteenwegen²

¹University of Antwerp, Belgium
²KU Leuven, Belgium

December 18, 2014

In this paper, the multiple travelling salesperson problem with hotel selection (m-TSPHS) is presented. This combinatorial optimisation problem is a generalisation of both the travelling salesperson problem with hotel selection (TSPHS) and the multiple travelling salesperson problem (m-TSP). As in the TSPHS, a tour is split into trips whose duration does not exceed a certain travel time, the objective of the m-TSPHS is to design a collection of tours, one for each available salesperson, while minimising the total travel costs. A set-partitioning formulation is presented as well as an efficient solution method together with a number of benchmark instances in order to evaluate the algorithm. Experimental results show that the proposed method is able to keep average gaps with respect to optimal solutions below 1.5%.

1 Introduction

In this paper, a generalisation of the travelling salesperson problem with hotel selection (TSPHS) (Vansteenwegen et al., 2011) is presented. In the TSPHS, a set of customers (or cities), a set of hotels and a maximum daily travel time are given. The objective is to design a minimum number of connected trips visiting all customers while minimising the total travelled time. Each trip should start from the hotel the previous trip ended at, and its duration should not exceed the maximum daily travel time.

The TSPHS has its roots in a real-life application (Vansteenwegen et al., 2010) in which a vehicle must travel through all the streets in a certain region while collecting geographical data. The objective is to minimise the number of days required to complete this task, while a hotel should be selected along the route for each night. This real-life application is called the mobile mapping van problem.

The multiple travelling salesperson problem with hotel selection (m-TSPHS) provides a generalisation of the TSPHS which considers a maximum number of working days, as well as multiple salespeople based in a central hotel (or depot). At the same time, the m-TSPHS is a generalisation of the multiple travelling salesperson problem (m-TSP) (Bektas, 2006). In the classical variant of the m-TSP, a set of salespeople is based at a single depot and the objective is to find a set of tours, each starting and ending at the depot, such that all customers are visited exactly once and the total cost of visiting the customers is minimised.

Like in the TSPHS, in the m-TSPHS, a distinction between the terms “trip” and “tour” is made. On the one hand, a trip corresponds to the work performed by a salesperson on a single day, and it is represented by a sequence of visits to customers, starting and arriving at the depot or one of the available hotels. A trip’s time duration is limited by a maximum daily travel time. On the other hand, a tour is the set of consecutive trips performed by a salesperson where the starting and ending point of the tour, i.e., the starting point and ending point of the first and last trip respectively, correspond to the depot. The number of trips in the tour is also bounded by a
maximum number of trips. In this problem, the objective is to find a set of tours, one for each salesperson, that, together, visit all customers while the overall travel cost is minimised.

The m-TSPHS has several practical applications which include the case of multiple employees or salespeople in a company who need to design their work trips, the programming of a fleet of trucks that have to travel long distances and need to split their entire journey into several trips, and the routing of electrical vehicles that need to recharge at one of the available stations.

The contributions of this paper are the following: the definition of a new variant of the m-TSP and the TSPHS along with a set-partitioning formulation, a procedure for generating benchmark instances with known optimal solutions as well as an efficient metaheuristic for the m-TSPHS from which a faster variant is obtained by slightly modifying its parameters.

The remainder of this paper is organised as follows. In Section 2, the relevant literature is reviewed and the similarities with and differences from related problems are discussed. In Section 3, a formal description of the problem is presented. Section 4 presents a local search algorithm as well as a metaheuristic for the m-TSPHS. In Section 5, a set of test instances is designed and a parametric analysis is performed. The results obtained by the solution method on the test instances are presented. Finally, conclusions and future research ideas are presented in Section 6.

2 Literature review

Since the m-TSPHS is a generalisation of the TSPHS (Vansteenwegen et al., 2011; Castro et al., 2013, 2014), which is, in turn, a generalisation of the classical travelling salesperson problem (TSP) (Applegate et al., 2007), it is related to many node routing problems that arise in the literature. In this section, similarities and differences between the m-TSPHS and those problems are discussed.

In the TSPHS, a single salesperson is considered. Therefore, all customers must be visited within a single tour. Furthermore, no limit is imposed on the number of trips and, as a consequence, a feasible solution can be found more easily at the expense of the number of required trips. In the m-TSPHS, the number of trips is limited by a maximum number of working days per tour and therefore it is not always trivial to find a feasible solution.

The m-TSPHS is also closely related to the following node routing problems:

- the multiple travelling salesperson problem (m-TSP) (Bektas, 2006),
- the vehicle routing problem (VRP) (Toth and Vigo, 2002),
- the multi-depot vehicle routing problem (MDVRP) (Cordeau et al., 1997; Polacek et al., 2004), and
- the multi-depot vehicle routing problem with inter-depot routes (MDVRPI) (Crevier et al., 2007).

As in the m-TSPHS, in each of these problems, a set of customers must be served by means of an available fleet of vehicles while minimising the travelled distance. Also, as in classical variants of the VRP, the fleet of vehicles is based at a depot. However, in those classical variants, a route cannot be split and must start and end at the depot, while, in the m-TSPHS, the required visits can be split into several trips, each starting and ending at the depot or one of the available hotels. In the MDVRP, multiple depots are considered. However, it is also impossible to split the workload, and for any given route, the starting and ending point is the same.

In routing problems with intermediate facilities (IFs), it is no longer required to start and end at a common depot. Therefore, in this class of problems, a tour can be split into trips which start and end at an IF. Several problems involving IFs can be found in the literature, some of which are

- the periodic vehicle routing problem with intermediate facilities (PVRP-IF) (Angelelli and Speranza, 2002),
- the waste collection vehicle routing problem with time windows (WCVRPTW) (Kim et al., 2006; Benjamin and Beasley, 2010) and node routing based solid waste collection problems (Hemmelmayr et al., 2013),
- the multiple-depot routing problem with inter-depot routes (MDVRPI) (Crevier et al., 2007),
- the capacitated arc routing problem with intermediate facilities (CARPIF) (Ghiani et al., 2001; Polacek et al., 2008), and
• the arc routing problem with intermediate facilities under capacity and length restrictions (CLARPIF) (Ghiani et al., 2004, 2010).

In these problems, like in the m-TSPHS, an IF can be visited several times. Despite their similarities, there are several differences between these problems with IFs and the m-TSPHS. The first one is that, in routing problems with IFs, a single trip is not bounded by a maximum length but by the vehicle capacity, whereas the vehicle in the m-TSPHS is implicitly assumed to have infinite capacity. The second difference is that, in problems like the MDVRPI and the CLARPIF, there is an upper bound on the total travelling time of each tour, but not on the travelling time of a single trip. In the m-TSPHS, the tour is only limited by its number of trips.

Recently, the electric vehicle routing problem with time windows and recharging stations (E-VRPTW) (Schneider et al., 2013) has been introduced. In this problem, the objectives are to minimise the number of used vehicles and the total travel cost. A trip between two recharging stations is limited by a power capacity, and the recharging time at each of the stations depends on the energy level.

Two problems which also deal with the hotel selection feature of the m-TSPHS are the orienteering problem with hotel selection (OPHS) (Divsalar et al., 2014) and the single salesperson problem with multiple time windows and hotel selection (Baltz et al., 2014) (TSPTWHS). The difference between the OPHS and the (m-)TSPHS is that, in the OPHS, the objective is to find a set of nodes to be visited that maximises the total collected score. That implies that it is possible that other nodes remain unvisited. In the (m-)TSPHS problem, all customers must be visited. On the other hand, TSPTWHS considers a single salesperson and time windows, while the problem tackled in this paper does not consider time windows but does involve multiple salespeople.

In summary, the more realistic problem tackled in this paper provides a generalisation of both the m-TSP and the TSPHS. In the same sense, the objective function differs from the one for the TSPHS in that it minimises the overall travel costs, while, in the TSPHS the primary objective is to minimise the number of trips, while, a secondary objective is to minimise the total tour duration.

Bektas (2006) discusses several variations on the m-TSP, including multiple depots, fixed or variable numbers of salespeople, fixed charges, and/or time windows. In the present paper, a new variant of the m-TSP is tackled. Next to the hotel selection part, it involves a fixed and known number of salespeople and no time windows.

3 Problem description

Given a set of \( s + 1 \) hotels \( \mathcal{H} = \{0, \ldots, s\} \) and a set of \( n \) customers \( \mathcal{C} = \{s + 1, \ldots, s + n\} \), the m-TSPHS is defined on a directed graph \( G = (\mathcal{V}, \mathcal{A}) \) where \( \mathcal{V} = \mathcal{H} \cup \mathcal{C} \) is the set of vertices in the graph and \( \mathcal{A} = \{(i,j) | i, j \in \mathcal{V}, i \neq j\} \) is the set of arcs in the graph.

A visiting time \( v_i \) is associated with each customer \( i \in \mathcal{C} \), with \( v_i = 0 \) for every hotel \( i \in \mathcal{H} \). Furthermore, the travel time \( r_{ij} \) as well as the travel cost \( c_{ij} \) associated with every arc \( (i,j) \in \mathcal{A} \) is known in advance.

Let \( x_{ij}^t \) be parameters which take the value 1 if location \( j \) is visited immediately after location \( i \) on trip \( t \), and 0 otherwise. A feasible trip \( t \) is a sequence of customers, represented by the vector \( (x_{ij}^t) \), such that:

(a) the trip starts and ends at one of the available hotels;

\[
\sum_{h \in \mathcal{H}} \sum_{(h,j) \in \mathcal{A}} x_{hj}^t = \sum_{h \in \mathcal{H}} \sum_{(i,h) \in \mathcal{A}} x_{ih}^t = 1
\]

(b) it does not contain an invalid sub-tour;

\[
\sum_{(i,j) \in \mathcal{A} : i,j \in S} x_{ij}^t \leq |S| - 1, \ \forall S \subseteq \mathcal{C}
\]

(c) its required time does not exceed a maximum value \( L \);

\[
\sum_{(i,j) \in \mathcal{A}} r_{ij} x_{ij}^t \leq L.
\]
Let $\mathcal{T}$ denote the set of all feasible trips, i.e., the set of trips for which conditions (a)–(c) are satisfied. Two parameters, $\phi_{ht}$ and $\lambda_{it}$, as well as a travel cost $\gamma_t$, are associated with every trip $t \in \mathcal{T}$. The travel cost is

$$\gamma_t = \sum_{(i,j) \in A} c_{ij} x_{t}^{ij}.$$ 

The parameters $\phi_{ht}$ and $\lambda_{it}$ are defined as follows:

$$\phi_{ht} = \begin{cases} 2 & \text{if trip } t \text{ starts and ends at hotel } h, \\ 1 & \text{if trip } t \text{ either starts or ends at hotel } h, \\ 0 & \text{otherwise}, \end{cases}$$

$$\lambda_{it} = \begin{cases} 1 & \text{if customer } i \text{ is visited by trip } t \\ 0 & \text{otherwise}. \end{cases}$$

Let $I$ be the set of trips which start and end at different hotels (i.e., the set of trips for which $\phi_{ht} = 1$), and denote by $\Delta(S)$ the set of trips that start and end in a specific subset $S$ of the hotels, i.e., $S \subseteq \mathcal{H}$. Finally, let $\Psi(S)$ be the set of trips that start at a hotel in $S$ and end at another hotel not in $S$, or vice versa.

Let $\mathcal{P}$ be the set of $m$ available salespeople. The objective is to find $m$ tours, each starting and ending at a hotel which is assumed to be identical and given ($i = 0$), i.e., the depot. The total duration should not exceed a predefined number of working days $D$.

Finally, let $y_{pt}$ be the binary variable which takes the value 1 if trip $t$ is assigned to salesperson $p$, and let the variable $w_{hp}$ denote the number of times the salesperson $p$ arrives at and departs from hotel $h$.

Using this notation, the m-TSPHS can be formulated as follows:

$$\min \sum_{p \in \mathcal{P}} \sum_{t \in \mathcal{T}} y_{pt}$$

s. t. \hspace{1cm} \sum_{p \in \mathcal{P}} \sum_{t \in \mathcal{T}} \lambda_{it} y_{pt} = 1 \quad \forall i \in \mathcal{C} \quad (2)$$

$$\sum_{t \in \mathcal{T}} y_{pt} \leq D \quad \forall p \in \mathcal{P} \quad (3)$$

$$\sum_{t \in I} \phi_{ht} y_{pt} = 2w_{hp} \quad \forall h \in \mathcal{H}, \forall p \in \mathcal{P} \quad (4)$$

$$\sum_{t \in \Delta(S)} y_{pt} \leq |\Delta(S)| \left( \sum_{t \in \Psi(S)} y_{pt} \right) \quad S \subseteq \mathcal{H} \setminus \{0\} \quad (5)$$

$$y_{pt} \in \{0,1\} \quad (6)$$

$$w_{hp} \in \mathbb{Z}^+ \quad (7)$$

Expression (1) considers the total cost. Constraints (2) ensure that all customers are visited by exactly one trip, while constraints (3) ensure that the number of trips assigned to each salesperson does not exceed the maximum allowed. Constraints (4) are the cardinality constraints for each hotel, while constraints (5) avoid invalid (sub)tours in the solution, i.e., tours that do not visit the starting hotel ($i = 0$). Finally, constraints (6) and (7) correspond to the integrality constraints.

Like in the TSPTWHS, a cost per stay at each of the hotels may be added to the objective function. Let $q_h$ denote the cost per stay at hotel $h$. The objective function (1) may then be modified to

$$\min \sum_{p \in \mathcal{P}} \left( \sum_{t \in \mathcal{T}} y_{pt} + \sum_{h \in \mathcal{H}} q_h w_{hp} \right)$$
to include hotel costs. Furthermore, a multi-depot variant may be defined by associating a different hotel \((h^p \in H)\) (or depot) with each salesperson. In that case, constraints (5) become:

\[
\sum_{t \in \Lambda(S)} y_{pt} \leq |\Lambda(S)| \left( \sum_{t \in \Phi(S)} y_{pt} \right) \quad S \subseteq H \setminus \{h^p\}, \quad p \in P.
\]

As noted above, the m-TSPHS is a generalisation of the classical TSP and, hence, it is also NP-hard. For this reason, we cannot expect to solve large instances to optimality. Therefore, there is a need for a heuristic solution procedure.

### 4 A metaheuristic for the m-TSPHS

Despite the similarities between the m-TSPHS and the other problems discussed in Section 2, heuristic or exact techniques tailored to the latter problems cannot be directly applied to the former. The reason of this is because they do not include moves or other elements that deal specifically with the hotel selection feature of this problem. The term “hotel selection” refers to the selection of the intermediate hotels to be used between two consecutive trips. Since the quality of a solution is strongly influenced by the selection of intermediate hotels (Kim et al., 2006; Castro et al., 2013; Divsalar et al., 2014), the inclusion of specific hotel selection operators is strongly required. In this section, a set of neighbourhood operators that have been proven to be effective in other solution methods for node-routing problems is implemented in an efficient metaheuristic for the m-TSPHS.

The metaheuristic proposed in this paper has been designed to iteratively perturb and improve a given solution. This heuristic is inspired by the behaviour of the iterated local search framework (ILS) (Lourenço et al., 2003, 2010). In an ILS, the search works on the domain of solutions reachable by a given local search algorithm. When a local optimum has been reached, the search is iterated by applying a typically random perturbation to the incumbent solution in order to move to a different part of the search space. The metaheuristic detailed in this section involves two different types of perturbation: one for the routing of the customers \((P_0)\), i.e., for their visiting order in the trips, and one for the hotel selection \((P_1)\) feature of the problem.

In Algorithm 1, the pseudo-code of the metaheuristic is presented. The algorithm starts by constructing Delaunay triangulation based neighbour lists (Reinelt, 1994) in order to speed up the evaluation of the neighbourhoods considered in the search. In the construction phase, the algorithm constructs an initial solution and improves it by means of the local search. The newly created solution is further improved in the improvement phase. At each iteration of the improvement phase, the incumbent solution \(X\) is perturbed first by means of operator \(P_0\), and subsequently improved by applying the local search. This is repeated as long as a better solution can be found. When it is no longer possible to improve the incumbent solution, it is perturbed using perturb operator \(P_1\) and another perturb-improve cycle is carried out. If the solution cannot be improved any more, the current iteration ends. Otherwise, the perturb operator is switched back to operator \(P_0\) and the search continues. The perturb operators \(P_0\) and \(P_1\) are detailed in Section 4.3.
Algorithm 1: Metaheuristic for the m-TSPHS

**require**
- Neighbourhood structures $N_k$, $k = 1, \ldots$
- Maximum number of iterations $i_{\text{max}}$
- Parameters $\theta_0$ and $\theta_1$

**initialisation**
- Build neighbour lists for each node
- Initialise counter: $i \leftarrow 0$

**construction phase**
- Construct initial solution $X$
- Initial improvement: $X \leftarrow \text{LOCALSEARCH}(X)$

**improvement phase**

while $i \leq i_{\text{max}}$ do
  Set perturb operator: $j \leftarrow 0$
  while $j \leq 1$ do
    Perturb: $X' \leftarrow P(X, \theta_j)$
    Improve: $X' \leftarrow \text{LOCALSEARCH}(X')$
    if $X'$ is better than $X$
      Move: $X \leftarrow X'$
      Reset perturb operator: $j \leftarrow 0$
    else
      Switch to the next perturb operator: $j \leftarrow j + 1$
  end while
  if no improvement has been made then
    $i \leftarrow i + 1$
  else
    $i \leftarrow 0$
  end if
end while
report $X$

**4.1 Initial solution**

In this section, two methods are presented to construct initial solutions. These methods are called C1 and C2. The construction algorithm generates one heuristic solution using each method and the best of both solutions is taken as initial solution for the improvement phase of the metaheuristic.

We define a heuristic solution $X$ to be a collection of trips $(X \subseteq T)$ which satisfy the conditions (a) and (b) but not necessarily condition (c) from Section 3. By dropping condition (c), trips which exceed the maximum travel time are also considered.

Neither of the two construction methods guarantees a feasible starting solution. Therefore, in order to decide whether one starting solution is better than another, the following criteria are used:

- if both solutions are feasible, the one with the best objective function value is selected,
- if only one solution is feasible, the infeasible one is dismissed, and,
- if neither of the two starting solutions are feasible, the one with the smallest value of $\tilde{F}$ is selected, where $\tilde{F}$ is the total time surplus with respect to the maximum trip duration $L$ with

$$\tilde{F}(X) = \sum_{t \in X} \left[ \max \left( \sum_{(i,j) \in A} r_g x^i_j - L, 0 \right) \right].$$  \hspace{1cm} (8)
4.1.1 Construction method C1

The first construction method is inspired by the insertion heuristic of Christofides et al. (1979) for the VRP. The method used for the m-TSPHS in this paper differs from the original one in that the number of tours is fixed in advance and corresponds to the number of salespeople.

Construction method C1 for the m-TSPHS works as follows. The \( m \) tours are built in parallel and customers are assigned to them without taking into consideration the time limit constraint. The criterion used to assign customers to the tours is the same as in the original implementation of Christofides et al. (1979), which is a metric based on the travel costs between the depot, a randomly selected seed customer, and the customer to be assigned to the tour.

When all customers have been routed, the tours are individually improved by using the operators 2-\textsc{opt}/\textsc{relocate} (see Section 4.2) simultaneously in a best improvement fashion.

Finally, each of the tours is split into feasible m-TSPHS trips by means of a tailored dynamic programming based algorithm for optimally partitioning a tour into feasible trips. This procedure has been adapted from the one in Prins (2004) for the VRP. If the resulting number of trips contained in any of the tours is larger than the maximum number of trips \( D \), a repair method iteratively removes customers from these tours and inserts them in the other tours in such a way that the total travel cost increases as little as possible. This procedure is repeated until the number of trips is small enough in each tour.

Construction method C1 generally results in starting solutions with a balanced number of customers in every tour.

4.1.2 Construction method C2

Unlike C1, the second construction method C2 constructs solutions with unbalanced numbers of customers per tour. This method is useful for instances in which good solutions involve one salesperson who performs most (or almost all) of the visits.

This method constructs a m-TSPHS solution from a TSP tour. The steps performed by this method are detailed below:

1. Given an instance with \( n \) customers and \( m \) salespeople, create \( m \) tours, that start and end at the central hotel \((i = 0)\) and do not visit any customer.

2. Assign the nearest non-visited customer (with respect to the depot) to each of the first \( m - 1 \) tours, so that all these tours visit exactly one customer.

3. Add the remaining non-visited customers to the last tour in a nearest neighbour fashion.

4. Improve the last tour by means of the Lin-Kernighan (Lin and Kernighan, 1973) heuristic and partition it into feasible m-TSPHS trips by using a splitting procedure.

5. If the resulting number of trips contained in the last tour is larger than the maximum number of trips \( D \), apply a the repair method as in construction method C1.

The splitting procedure in step 4 is adapted from the one in Prins (2004). In its original implementation, VRP routes are created from a giant tour provided the total travel distance is minimised and each of the routes is feasible with respect to the capacity constraints. In our implementation, trips are bounded by the maximum trip duration \( L \) and considers multiple hotels instead of a common depot.
4.2 Neighbourhood operators

Several operators have been implemented, some of which are well known operators which have been successfully applied in solution methods for related problems. Other operators we have implemented are problem specific. In the figures used to illustrate the neighbourhood operators implemented in this section, white squares represent hotels while black circles represent customers.

On the one hand, the well-known operators 2-opt (Croes, 1958), Relocate, Exchange and Cross (Laporte et al., 2000) are used to improve a solution by manipulating the visiting order of the customers, i.e., by swapping, flipping and reallocating chains of consecutive customers (node-routing). On the other hand, the operator ChangeHotel introduced by Castro et al. (2013), and the new operators RemoveHotel and InsertHotel deal with the hotel selection feature of the problem.

Operator 2-opt (see Figure 1) attempts to improve a solution by removing two arcs of a trip and replacing them with two new arcs. This implies the reversal of the visiting order in a segment of the trip. Operator Relocate (see Figure 2) relocates (and flips, if necessary) a chain of up to three consecutive customers anywhere in the solution, either within the same trip, or across different trips.

Operator Exchange (see Figure 3) swaps two chains of up to three consecutive customers from one trip to another. Operator Cross (see Figure 4) swaps the final parts of two different trips that share the same ending hotel with respect to a crossing point.

Operator ChangeHotel (see Figure 5) changes the selection of an intermediate hotel between two consecutive trips. Operator RemoveHotel (see Figure 6) joins two consecutive trips by removing the intermediate hotel.
between them. This implies that the number of trips is decreased by one. Operator \texttt{INSERTHOTEL} (see Figure 7) splits a trip by inserting an intermediate hotel. This can be seen as the opposite move of \texttt{REMOVEHOTEL}. Analogously, this operator implies that the number of trips is increased by one.

4.3 Perturb operators \( P_0 \) and \( P_1 \)

The first perturb operator introduces a perturbation of the routing of the customers. Given a solution \( X \), operator \( P_0 \) relocates \( r_0 \) customers, where \( r_0 = \lceil n \theta_0 \rceil \), \( n \) is the total number of customers and \( \theta_0 \) is a parameter in the interval \((0, 1]\). Which \( r_0 \) customers are relocated is determined at random. The operator removes each of the \( r_0 \) customers from its current position, and reinserts it at another randomly selected location. This is repeated until the \( r_0 \) customers have been relocated.

The second perturb operator introduces a perturbation of the hotel selection. As noted in Castro et al. (2013) for the TSPHS and in Kim et al. (2006) for the WCVRPTW, a good selection of intermediate hotels in the tours has a major impact on the quality of the final solution. This pleads for an operator that diversifies the search in the domain of the hotel selection. To this end, the operator \( P_1 \) randomly changes the selection of \( r_1 \) intermediate hotels in a given solution \( X \). The number \( r_1 \) depends on the number of intermediate hotels.

Let \( D_p \) be the number of trips contained in tour \( p \) and let \( m \) again be the total number of tours. Then, the total number of intermediate hotels \( D_{\text{int}} \) is calculated as follows:

\[
D_{\text{int}} = \sum_{p=1}^{m} D_p,
\]
this way, \( r_i = \frac{D_i}{\text{int} \theta_i} \) where \( \theta_i \) is a parameter that takes a value in the interval \((0, 1]\). The selection of the \( r_i \) hotels to be changed is made at random. The operator removes each of the selected hotels from its tour and replaces it with a randomly selected hotel.

### 4.4 Local search

In this section, the details of a local search for the m-TSPHS are presented. The local search plays a central role in the metaheuristic described at the beginning of Section 4 and shown in Algorithm 1. The local search combines a steepest descent with a strategic oscillation.

Given a solution \( X \), the local search tries to minimise the function

\[
F(X, \omega) = \sum_{t \in X} c_{ij}^t + \omega \tilde{F}(X),
\]

where \( \omega \) is a penalty factor and \( \tilde{F}(X) \) measures the extent to which the solution \( X \) is infeasible (see Eq. 8). If \( X \) is feasible, then \( \tilde{F}(X) = 0 \). Otherwise, \( \tilde{F}(X) > 0 \). The relaxation of the condition (c) in combination with a penalty for a violation is a very powerful idea to integrate diversification in the search. This diversification is known as *strategic oscillation* (see Section 4.4.2). In Algorithm 2, the steps performed by the local search are presented.

#### Algorithm 2 Local search with strategic oscillation

**input:** Solution \( X 

**procedure** LOCALSEARCH(X)

  initialise
  
  | Penalisation: \( \omega \leftarrow 0 \) |
  | Best solutions: |
  | if \( X \) is feasible |
  | \( X^* \leftarrow X \) |
  | else |
  | \( X^* \leftarrow \emptyset \) |
  | \( \hat{X} \leftarrow X \) |

  repeat
  
  | Apply descent: DESCENT(X, \( \omega \)) |
  | if \( X \) is feasible then |
  | if \( X \) is better than \( X^* \) then |
  | Update best feasible: \( X^* \leftarrow X \) |
  | Reset penalty: \( \omega \leftarrow 0 \) |
  | else |
  | if \( \tilde{F}(X) < \tilde{F}(\hat{X}) \) then |
  | Update best infeasible: \( \hat{X} \leftarrow X \) |
  | if \( \omega > 0 \) then |
  | \( \omega \leftarrow 2\omega \) |
  | else |
  | \( \omega \leftarrow 2^{-6} \) |

  while stopping criteria not met

  report \( X^* \) if \( X^* \neq \emptyset \); \( \hat{X} \) otherwise

end procedure
4.4.1 Descent

The basic building block of the local search is a steepest descent method. The descent method implemented in this paper works with the neighbourhoods resulting from the operators defined in Section 4.2. At each iteration, all possible moves contained in every neighbourhood are evaluated and the one that results in the largest objective function decrease is selected. Algorithm 3 shows the pseudo-code of the descent structure. The neighbourhood structures \( N_1, \ldots, N_7 \) correspond to the operators 2-opt, Relocate, Exchange, Cross, ChangeHotel, InsertHotel and RemoveHotel, respectively.

**Algorithm 3 Steepest descent**

**Input:** Solution \( X \), penalty factor \( \omega \), and neighbourhoods \( N_k \)

**Procedure** \( \text{Descent}(X, \omega) \)

repeat

Find best neighbour: \( X' \leftarrow \arg \min_{X \in \bigcup \{N_k(X)\}} F(X', \omega) \)

Evaluate: if \( F(X', \omega) < F(X, \omega) \) then

Move: \( X \leftarrow X' \)

while a better solution has been found

end procedure

4.4.2 Strategic oscillation

Strategic oscillation is a mechanism that enables the search to switch between feasible and infeasible parts of the search space in order to integrate diversification in a local search algorithm (see Brandão and Mercer (1997); Brandão (2011); Gallego et al. (2013); Gendreau et al. (1994, 2006); Lozano et al. (2013); Palhazí-Cuervo et al. (2014)).

In Glover and Hao (2011), several forms of strategic oscillation are described. The one adopted in this paper is an oscillation over parameter settings. As shown in Eq. (9), a penalty factor \( \omega \) is attached to the violation of trip duration constraints. This factor is dynamically updated during the search: a small value of \( \omega \) leads the search towards the infeasible portions of the search space, while a large value of \( \omega \) moves the search towards feasible solutions. This is referred to as single parameter oscillation.

The oscillation strategy works as follows. At any iteration of the local search, given a current solution \( X \) and a value of \( \omega \), a local optimum is obtained by applying the descent method \( (X \leftarrow \text{Descent}(X, \omega)) \). The algorithm then proceeds as follows:

1. If the newly obtained solution lies in the feasible portion of the search space and the stopping conditions are not yet met, then the search is intentionally guided towards the infeasible portion of the search space. To this end, the penalty factor is set to zero.
2. If the newly obtained solution lies in the infeasible portion of the search space and the stopping conditions are not yet met, the search is intentionally guided towards the feasible portion of the search space. To this end, the penalty factor is set to a non-zero value, and is iteratively increased.

Generally, when the penalty factor has been set to zero, the produced local optimum lies in the infeasible portion of the search space, so that feasibility has to be recovered by increasing the penalty factor iteration by iteration.

The oscillation strategy can be seen as a way to introduce a perturbation in the current solution. Despite the fact that the local search is a purely deterministic algorithm, it might still be possible, due to the oscillation strategy, that the feasibility is recovered in a different region of the search space and, hence, new solutions may be explored.

The local search itself does not guarantee a feasible solution. Even when the value of \( \omega \) is large, it might not be possible to find a sequence of moves that makes a solution feasible. Therefore, the descent terminates when any of the two following stopping criteria are met:
1. the value of $\omega$ is greater than $\omega_{\text{max}}$ ($\omega_{\text{max}} = 2^{15}$ in our implementation) or,
2. a feasible solution has been found by the Descent method, but it is not better than the globally best feasible solution $X^*$.

5 Computational experiments and results

5.1 Test instances

In order to test the algorithm developed in the previous section, a procedure to generate test instances has been designed. The instances are created in such way that the optimal solution is known. The procedure for generating instances is as follows:

1. Specify the number of customers, $n$, and a number of salespeople, $m$.
2. Create an instance for the m-TSP by generating a set of $n + 1$ random 2D points within a square of 400 units of length. These points represent a central hotel (node 0) and $n$ customers.
3. Solve the m-TSP to optimality.
4. Set all service times to zero.
5. Given the optimal solution for the m-TSP, and a maximum number of trips $M_T$, insert artificial hotels along each of the $m$ tours in such a way that the optimal cost for the m-TSPHS is the same as for the m-TSP. For each of the $m$ tours, perform the following:
   a) Let $l_m$ be the length of the $m$th tour and let $l_m(u, v)$ denote the accumulated length of tour $m$, from node $u$ to node $v$, inclusive. Initialise the trip length $\lambda$ by setting it to $l_m/M_T$.
   b) Start from node 0, and follow the tour either until reaching the first node $v$ for which $l_m(0, v) \geq \lambda$ or until reaching node 0, whichever happens first. If $v$ is different from node 0, then create an artificial hotel at the location of node $v$. Update $\lambda = l_m(0, v)$.
   c) Continue as in the previous step, but starting from node $v$ instead of from node 0. Repeat the step until the end of the tour has been reached.
6. Take the length of the longest trip in the instance and use it as the maximum trip duration $L$.
7. Take the tour with the largest number of trips, and set the parameter $D$ to its number of trips.
8. Insert additional hotels at random positions until the total number of hotels is $2M_T$.

With this procedure, at most $M_T - 1$ artificial hotels are created. Finally, two types of instances are considered: instances with a fully random layout and instances with a clustered layout. In Section 5.3, a small set of test instances is generated by using this procedure in order to fine tune the parameters of the algorithm, while, in Section 5.4, a larger set of instances is generated in order to test the metaheuristic’s performance.

5.2 Implementation details

The metaheuristic was implemented in C/C++ and compiled using GCC 4.8.2. All experiments were run on an Intel Core i7-870 processor (2.93 GHz) and 4 GiB of RAM running Arch Linux.

The Lin-Kernighan heuristic used in construction method C2 (see Section 4.1.2) corresponds to the one implemented by Applegate et al. (2006).


5.3 Parametric analysis

In this section, the details of a statistically designed experiment to fine tune the parameters of the metaheuristic described in Section 4 are presented. The aim of the experiment is to determine the most robust parameter configuration to maximise the quality of the solutions produced by the algorithm for a broad range of instances. The methodology is similar to the one used in Castro et al. (2013). A full factorial experiment is conducted and the results are analysed by estimating two mixed-effects analysis of variance (ANOVA) models.

For the first ANOVA model, the function $F(X, \omega)$ (introduced in Section 4.4) is used as performance measure while, for the second model, the computation time is used as performance measure. When doing so, a large value is used for the penalty factor $\omega$, namely 100. This makes it possible to determine the levels of the algorithm’s parameters $i_{\text{max}}, \theta_0$, and $\theta_1$ that maximise the quality of the solutions and, at the same time, minimise the occurrence of infeasible solutions.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of customers ($n$)</td>
<td>50, 100, 200</td>
</tr>
<tr>
<td>Number of salespeople ($m$)</td>
<td>2, 3, 4, 5, 6, 7, 8, 9, 10</td>
</tr>
<tr>
<td>Maximum number of trips ($M_T$)</td>
<td>5, 10</td>
</tr>
</tbody>
</table>

Table 1: Parameters used to generate instances for the parametric analysis

For the experiment, a total of 60 instances, 30 for each type (random and clustered), has been generated using the procedure described above and random combinations of parameter values shown in Table 1. In Table 2, the metaheuristic’s parameters and the tested levels are presented. For each combination of the metaheuristic’s parameter levels and for each instance, the algorithm is executed 10 times, producing $3 \times 11 \times 6 \times 60 \times 10 = 118800$ observations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Tested levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_{\text{max}}$</td>
<td>10, 20, 30</td>
</tr>
<tr>
<td>$\theta_0$</td>
<td>0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.60, 0.80, 1.00</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>0.10, 0.20, 0.40, 0.60, 0.80, 1.00</td>
</tr>
</tbody>
</table>

Table 2: Parameter levels tested in the experiment

The results of the first ANOVA model are as follows: all the main effects as well as the interaction $\theta_0 \times \theta_1$ turned out to be significant. The statistical details of the estimated ANOVA model are presented in Table 3 and have been obtained by means of the statistical package R (R Core Team, 2014).

<table>
<thead>
<tr>
<th>Source</th>
<th>numDF</th>
<th>denDF</th>
<th>F-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_{\text{max}}$</td>
<td>2</td>
<td>118673</td>
<td>134.4473</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>$\theta_0$</td>
<td>10</td>
<td>118673</td>
<td>11.5457</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>5</td>
<td>118673</td>
<td>105.4247</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>$\theta_0 \times \theta_1$</td>
<td>50</td>
<td>118673</td>
<td>1.4558</td>
<td>0.0194</td>
</tr>
</tbody>
</table>

Table 3: Significant effects

Figure 8 shows the least-squares mean plots in order to tune the parameter $i_{\text{max}}$ of the metaheuristic (Figure 8a) and the parameters $\theta_0$ and $\theta_1$ (Figure 8b). From Figure 8a, it is clear that the quality of a solution increases with $i_{\text{max}}$. Figure 8b shows that parameter $\theta_0$ should not be larger than 0.35. This interaction plot also shows that, for lower levels of $\theta_1$, it is important that $\theta_0$ is in the interval [0.15, 0.25], but, for larger levels of $\theta_1$, values of $\theta_0$
smaller than 0.35 suffice to obtain good solutions. Finally, it is possible to see that the best solutions are obtained with $\theta_1 = 1$. Based on these observations, the algorithm’s parameters have been set to the values shown in Table 4: $i_{\text{max}}$ is set to 30 while $\theta_0$ and $\theta_1$ are set to 0.2 and 1.0, respectively.

![Figure 8: Least-squares mean plots](image)

The interpretation of the chosen levels for $\theta_0$ and $\theta_1$ is as follows: 20% of the customers are randomly relocated every time operator $P_0$ is applied, while all intermediate hotels are randomly modified when operator $P_1$ is applied. This means that whenever it is not possible to find good solutions by means of perturb-improve cycles using the perturb operator $P_0$ in Algorithm 1, an entirely new hotel selection sequence is tested as a result of applying operator $P_1$. Note that, however, this is not equivalent to a random restart since none of the customers is moved and, therefore, good sequences of arcs are carried over from one solution to another.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_{\text{max}}$</td>
<td>30</td>
</tr>
<tr>
<td>$\theta_0$</td>
<td>0.20</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 4: Selected parameter settings

5.4 Results

In this section, the computational results achieved by the metaheuristic proposed in this paper on a set of randomly generated instances are reported. The different parameters levels used to generate instances are presented in Table 5. For each type of instance and for each parameter combination, five different instances have been generated to produce a total of 900 instances. The data shown in the remainder of this section corresponds to the average of 10 executions of the algorithm on each of the instances.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of customers ($n$)</td>
<td>50, 75, 100, 200, 300</td>
</tr>
<tr>
<td>Number of salespeople ($m$)</td>
<td>2, 3, 4, 5, 6, 7, 8, 9, 10</td>
</tr>
<tr>
<td>Maximum number of trips ($M_T$)</td>
<td>5, 10</td>
</tr>
</tbody>
</table>

Table 5: Parameters used to generate instances for the tests
Table 6 contains the average gaps produced by the metaheuristic with respect to optimal solutions. The table also shows that all average gaps with respect to the optimal solutions lie between $0.0\%$ ($n = 50, m = 2$) and $1.4\%$ ($n = 300, m = 6$). Figure 9 shows that the average percentage gap increases with the size of the instance and that, in general, the average gaps turned to be higher for instances with a fully random layout.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.0</td>
<td>0.1</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>75</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>100</td>
<td>0.6</td>
<td>0.1</td>
<td>0.2</td>
<td>0.5</td>
<td>0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>200</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.5</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.6</td>
<td>0.9</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>300</td>
<td>1.2</td>
<td>0.9</td>
<td>1.0</td>
<td>1.1</td>
<td>1.4</td>
<td>1.3</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Total</td>
<td>0.5</td>
<td>0.4</td>
<td>0.4</td>
<td>0.5</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 6: Average gaps (%)

Figure 9: Average gaps (%) for instances with a random layout and with a clustered layout

Table 7 shows the percentage of optimal solutions found by the metaheuristic. From this table, it is possible to see that optimal solutions were found in $52\%$ of the cases. Of the easiest instances ($n = 50, m = 2$), the metaheuristic solved $96\%$ to optimality. However, for larger instances, this percentage drops to $1.5\%$ ($n = 300, m = 10$).

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>96.5</td>
<td>94.5</td>
<td>85.0</td>
<td>80.0</td>
<td>87.0</td>
<td>86.0</td>
<td>71.0</td>
<td>86.5</td>
<td>77.0</td>
<td>84.8</td>
<td>84.8</td>
</tr>
<tr>
<td>75</td>
<td>79.0</td>
<td>87.0</td>
<td>68.0</td>
<td>77.0</td>
<td>85.0</td>
<td>81.0</td>
<td>85.5</td>
<td>74.0</td>
<td>69.5</td>
<td>78.4</td>
<td>78.4</td>
</tr>
<tr>
<td>100</td>
<td>73.0</td>
<td>69.5</td>
<td>68.0</td>
<td>51.0</td>
<td>53.0</td>
<td>65.5</td>
<td>61.0</td>
<td>58.5</td>
<td>48.5</td>
<td>60.9</td>
<td>60.9</td>
</tr>
<tr>
<td>200</td>
<td>29.5</td>
<td>28.5</td>
<td>14.5</td>
<td>30.5</td>
<td>7.5</td>
<td>19.5</td>
<td>24.5</td>
<td>33.5</td>
<td>19.5</td>
<td>23.1</td>
<td>23.1</td>
</tr>
<tr>
<td>300</td>
<td>21.0</td>
<td>13.5</td>
<td>11.5</td>
<td>24.5</td>
<td>6.0</td>
<td>16.5</td>
<td>12.5</td>
<td>8.5</td>
<td>1.5</td>
<td>12.8</td>
<td>12.8</td>
</tr>
<tr>
<td>Total</td>
<td>59.8</td>
<td>58.6</td>
<td>49.4</td>
<td>52.6</td>
<td>47.7</td>
<td>53.7</td>
<td>50.9</td>
<td>52.2</td>
<td>43.2</td>
<td>52.0</td>
<td>52.0</td>
</tr>
</tbody>
</table>

Table 7: Percentage of optimal solutions found
In our results, there is no clear indication that, in general, the instances become harder as the number of salespeople increases. This can be seen in Figure 10 where two plots are shown. Figure 10a shows a plot with the average gaps for each number of salespeople and type of instance, while, Figure 10b contains a plot showing the percentage of optimal solutions found for each number of salespeople and per type of instance.

Finally, Table 8 contains the average computing times required by the metaheuristic. The average times for the smallest and the largest instances are 1.1 s and 31.8 s, respectively. The computation time does not vary substantially with the number of people.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50</td>
<td>0.8</td>
<td>0.9</td>
<td>1.2</td>
<td>1.1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.2</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>1.8</td>
<td>1.9</td>
<td>1.9</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.1</td>
<td>2.3</td>
<td>2.2</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>2.8</td>
<td>2.8</td>
<td>3.3</td>
<td>3.5</td>
<td>3.6</td>
<td>3.4</td>
<td>3.5</td>
<td>3.8</td>
<td>3.9</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>14.4</td>
<td>14.0</td>
<td>13.0</td>
<td>12.9</td>
<td>13.3</td>
<td>13.3</td>
<td>12.6</td>
<td>12.3</td>
<td>13.1</td>
<td>13.2</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>33.4</td>
<td>33.4</td>
<td>30.5</td>
<td>32.4</td>
<td>31.1</td>
<td>33.7</td>
<td>30.3</td>
<td>29.8</td>
<td>31.4</td>
<td>31.8</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>10.7</td>
<td>10.6</td>
<td>10.0</td>
<td>10.4</td>
<td>10.2</td>
<td>10.7</td>
<td>9.9</td>
<td>9.9</td>
<td>10.4</td>
<td>10.3</td>
</tr>
</tbody>
</table>

Table 8: Average computation times (s)

A different round of tests was executed by disabling the \( P_1 \) perturb operator and, hence, reducing the algorithm to a regular ILS which only shakes the routing of the customers when a local optimum has been found by the local search. The results of this extra experiment are as follows. Firstly, the percentage of optimal solutions drops from 52.0% to 44.9% on average. Secondly, the overall average gap increases from 0.5% to 1.2%. On the positive side, the computational time decreases from 10.3 to 5.6 s. It could be concluded that disabling the \( P_1 \) perturbation leads to a very fast algorithm with only a minor loss in solution quality. It should be noted, however, that the algorithm is more robust if the \( P_1 \) operator is enabled since the standard deviation of the gap is 5.1% if \( P_1 \) is disabled and 1.0% otherwise. Considering quality of the solutions, the results of this second experiment back up the way the metaheuristic was conceived: introducing stochastic elements for each of the characteristic of the problem, the routing and the hotel selection, has a beneficial impact on the results.
6 Conclusions

The multiple travelling salesperson problem with hotel selection is a recent variant of the classical travelling salesperson problem. It has several practical applications which include the case of multiple employees or salespeople in a company that need to design their work trips, the programming of a fleet of trucks that have to travel long distances and need to split the entire journey into several days, the routing of electrical vehicles that need to recharge at one of the available stations, etc.

In this paper, a mathematical formulation for the m-TSPHS is introduced. The formulation can easily be adapted to include features such as hotel costs and multiple depots. A procedure to create test instances with known optimal solutions has been designed too, as well as a metaheuristic to solve the m-TSPHS. This metaheuristic exploits well-known neighbourhoods for node-routing problems and specific hotel selection neighbourhoods. A parametric analysis to fine tune our approach has also been detailed. The solution algorithm is shown to be robust since it is able to keep the average gaps below 1.5% for the largest tested instances, with a standard deviation of 1.0%. The metaheuristic is able to find the optimal solution, on average, in 52.0% of the cases within computational times of 10.3 s on average. A fast variant of the algorithm is obtained by disabling the hotel selection perturb operator.

Further research could focus on, on the one hand, on a more realistic version of the problem, including time windows, lunch breaks, salaries, etc., and, on the other hand, on the design of parallel and/or cooperative solution methods in order to solve large scale instances.

Acknowledgement

The first three authors thank the Fonds voor Wetenschappelijk Onderzoek - Vlaanderen (FWO) as well as Combinatorial Optimization: Metaheuristics and EXact methods (COMEX) project for financial support.

References


