The Single-Double Balanced Traveling Salesman Problem

Tatiana Bassetto - Francesco Mason

Dipartimento di Matematica Applicata
Università degli Studi Ca' Foscari di Venezia

Abstract. In the Single-Double Balanced Traveling Salesman Problem (SDB-TSP), the customers must be visited over a period of two days: some must be visited daily, and the others on alternate days (even or odd days). The salesman’s objective is to minimize the total distance travelled over the two tours; moreover, the number of customers visited in every tour must be ‘balanced’, i.e. the difference between the maximum and the minimum number of visited customers must be less than a given threshold. Although this problem may be viewed as a particular case of the Period Traveling Salesman Problem, in the SDB-TSP the assumptions allow for emphasising on routing aspects, more than on the best assignment of the customers to the various days of the period. The paper proposes a problem settlement and two (fast) heuristic algorithms. In order to enhance the properties of the proposed procedures, an application is also given.

Keywords: period routing problem, period traveling salesman problem, logistic, heuristic algorithm.

Introduction

The purpose of this paper is to present heuristic techniques to solve the SDB-TSP, the Single-Double Balanced Traveling Salesman Problem.

In the organization of picking up orders for commercial firms, it often happens that an agent (or vehicle) has to visit his/her customers (or cities) periodically but at different times on the grounds of the number and the frequency of the orders. One of the simplest cases is when customers are divided into two sets over a period of two days: customers visited daily and customers visited only once in the period. Moreover, the agent wish to work every day almost the same amount: this way, some balancing constraints must be imposed on the two tours.

The problem can be defined on a complete, undirected and weighted graph $G = (V, E)$, where $V = \{v_0, v_1, \ldots, v_n\}$ is the node set and $E = \{(v_i, v_j): v_i, v_j \in V, i \neq j\}$ is the edge set. The nodes in $V$ set correspond to customers (or cities), including the home city (or the depot) $v_0$, from which the salesman (or the vehicle) starts routes. Weights are imposed on the arcs: $c_{ij}$ represents the length or cost of the edge $(i, j)$. Moreover, a value $g^*$ is given, which represents the maximum number of customer which can be visited on each day (‘balancing’ constraint).

The SDB-TSP may be viewed as a particular case of the Period TSP (PTSP) [4]. This last problem generalizes the TSP by extending the planning period to $p$-days. In PTSP every customer must be visited a specified number of times: for each customer a set of feasible (allowable) combinations of visit days is also given. The aim is to build $p$ routes (one route for every day) in order that the distance travelled over the entire period is minimized.
In [5] a constraint on the maximum length of each route is also given; in fact, in the subsequent literature this ‘balance constraint’, to our knowledge, is no more considered as such in the solution procedure.

The number of days of the period in the SDB-TSP is two \((p = 2)\) and customers must be visited once or twice. This way, it is clear that in the SDB-TSP the node set \(V\) is partitioned into two subsets: \(S\) (Single) is the set of the customers that must be visited on alternate days and \(D\) (Double) is the set of the customers that must be visited daily. It is clear that, differently from the PTSP, in the SDB-TSP there is not a set of different combinations of visit days; moreover, in the SDB-TSP we impose the following ‘balance constraint’: a maximum number of customer which can be visited on each day.

Obviously, if the number of customer which are visited on the even days must be the same or can differ at most by one unit with respect to the odd days, the number of single-nodes (customers), which are visited in every tour, must be equal to \(\lfloor |S|/2 \rfloor\).

The classical TSP has been studied extensively over the last 50 years: a fundamental work is the book by Lawler et al. [10], but see also Laporte [9].

The Period Routing Problem by Christofides and Beasley in 1984 [5] is the first published paper that develops solution approaches for the routing problem over more than one day. The purpose of the authors is to study the PVRP (Period Vehicle Routing Problem), a problem with weighted-nodes and a capacity constraint on the vehicles: the authors give an exact formulation of the problem, but they solve it through heuristic algorithms, since PTSP is NP-hard. The solution of the problem is obtained through the relaxation of the PVRP for each day of the period. They reduce it to two subproblems: a median problem, in order to determine customers to visit in every day of the period, and a TSP for every day, in order to minimize the total distance travelled. They claim to expect that minimizing the sum of the lengths of the salesman tour for each day would also tend to minimize the total length of the PVRP.

In 1992 Paletta develops two heuristic techniques for solving the PTSP [11]. Both heuristics use the following structure: first, they assign a non-visited customer to an allowable day combination (weights are associated with every customer); second, a TSP on every day is solved by a dynamic programming technique. The weights are recalculated according to the distance from the unassigned customers to each day’s tour: this procedure is repeated until all customers are assigned. In a third step, improvement techniques, like the 2-opt improvement, are applied to every tour.

Chao et al. [4] in 1992 present a new heuristic technique for solving the PTSP (quoted to be ‘the best approach’ in Cordeau et al. [6]).

In [4] an initial solution is built by an “inizialization” step, in order to obtain a feasible solution: in this step the objective is to minimize the maximum number of customers visited in a day, in order to start with an initial solution that “balances” the number of customers which are visited on each day. Then, an improvement step is employed using Dueck’s method (2-opt improvement), which is followed by a re-initialization. The heuristic changes the assigned allowable day combination of customers in order to minimize the total travelled distance. We want to underline that the initial goal to balance the number of customers can not be maintained with the improvement steps: it can happen that in the best solution reached by the algorithm the tours visit a significantly different number of customers on each day of the period.

Cordeau, Gendreau and Laporte [6] in 1997 present a tabu search heuristic for the solution of three period routing problems: the PVRP, the PTSP and the PVRP with more depots. (The algorithm is unique: some parameters take account of the different three problems).
In a first phase, nodes, that are represented by Euclidean coordinates, are sorted “in increasing order of the angle that they make with the depot” [6] and a visit combination is randomly chosen. The algorithm assigns customers to the routes, but the route assigned to the last vehicle may be infeasible. In order to reach a feasible solution GENI heuristic is used to remove and replace nodes. Then, the tabu search heuristic uses some improvement techniques like 4-opt and the cheapest-insertion techniques. This algorithm has the advantage to use a very small number of user-controlled parameters. Computational results that are tested on literature instances show that the proposed heuristic is better than the previous ones in all the three types of problems. We want to underline that the maximum length constraint of each route is not dealt with by the authors in the heuristic. This way, (as foreseen) the best solution (obtained by the algorithm) can be the one in which tours visit a really different number of customer in a day.

In 2002 Paletta [12] presents a new heuristic algorithm for the PTSP. Starting from the depot, a tour is built inserting one customer at time through a heuristic insertion rule. If a tour is empty (i.e. if one day of the planning period is not used: infeasible solution), at least one customer is removed (through an explicit rule) until the travelling salesman must visit at least one customer every day. When a feasible solution is reached, an improvement procedure occurs to test if it is possible to improve the previously reached feasible solution in order to minimize the total length travelled by the vehicle. The author gives also a stopping rule in order to limit calculations. The proposed heuristic was tested on some instances taken from the literature and computational results indicate that, in many instances, the new heuristic finds a larger number of optimal solutions.

Also Baptista et al. [2], studying the collection of recycling paper containers in Portugal, claim to resolve a particular case of the Period VRP, but their problem is effectively involved with stochastic frequency of visit, and it is quite different from the deterministic problems quoted from the preceding literature.

Bertazzi, Paletta and Speranza [3] in 2004 propose “an improved heuristic for the PTSP”, i.e. an improved heuristic with respect to the one proposed by Paletta in [12]. The heuristic is a construction type algorithm consisting in the following steps: (i) a tour construction phase to build the tours, which selects a not yet processed city (here it can be generated an unfeasible solution); (ii) a feasibility procedure to get a feasible solution (this phase occurs only if it is necessary), (iii) an improvement procedure to modify and improve the previous obtained feasible solutions. This heuristic introduces new improvement procedures with respect to [12] and computational results indicate it to be better than the previous ones. We observe that the additional instances, which are variants of some of the classical instances (the coordinates of some nodes of the instances are changed), in order to be rigorous, can not be directly compared with previous instances and can not be summed up in their statistics to the other instances.

Last year (2005) Ghiani et al. [7] present a heuristic for the undirected periodic rural postman problem, in which every edge of the graph must be visited a given number of times over a planning period of p-days and service days must be equally spaced.

As aforesaid, the SDB-TSP is a particular case of the PTSP in which the period consists of two days and, in this way, two tours T1* and T2* must be built. Customers can be only of two types: the ones to visit daily (D) and to insert in every tour, and the others which must be visited in one of the two days (S) and inserted in only one of the two tours (in order to visit them on alternate days).

The objective is to minimize the total distance travelled over the two tours.

Moreover, we wish to balance the number of the cities among the tours. This goal can be achieved in different ways: by a multi criteria formulation, by introducing suitable constraints or by an objective function that includes travel costs.
1. An integer programming model for the SDB-TSP.

Let $G = (V, E)$ a complete graph of $n$ nodes ($n > 1$) without loops. Let $c_{ij}$ be the weight of the edge $(i, j)$. In the SDB-TSP the set $V$ can be partitioned into three (disjoint) subsets:

- the depot $\{v_0\}$;
- the set of single-nodes $S = \{s_1, s_2, \ldots, s_k\}$, i.e. the ones to visit once over two days;
- the set of double-nodes $D = \{d_1, d_2, \ldots, d_h\}$, i.e. the ones to visit every day.

Obviously, $h + k = n - 1$.

In what follows, a node belonging to set or tour $X$, is called $X$-node.

We want to build two tours, $T_1$ and $T_2$ (one for every day), which satisfy a balance constraint, in order to minimize the total travelled distance. Both tours visit the depot $v_0$ and all the $D$-nodes, while every single-node, i.e. every $S$-node, can be inserted only in one of the two tours, $T_1$ or $T_2$.

This way, in every feasible solution, $S$ is partitioned into two subsets, $S^1$ and $S^2$ (with $S = S^1 \cup S^2$ and $S^1 \cap S^2 = \emptyset$), the first of one made up of nodes visited on the first (or odd) day and the second one on the second (or even) day.

Let $k_1$ be the cardinality of $S^1$ and $k_2$ the cardinality of $S^2$, ($k_1 + k_2 = k$). $T_1$-nodes constitute the set $\{v_0\} \cup D \cup S^1$, while $T_2$-nodes are $\{v_0\} \cup D \cup S^2$.

Let $g^*$ be the value of the maximum number of customer visited each day: in a feasible solution we have the following constraints

$$k_1 + h \leq g^*, \quad k_2 + h \leq g^*.$$

The problem can be formulated as an integer linear programming one introducing the following Boolean variables:

- $x_{ijq} = 1$ if and only if the customer $j$ is visited immediately after $i$ on the $q$-day and 0 otherwise, ($q = 1$ or $2$);
- $y_{iq} = 1$ if the customer $i$ is visited on the $q$-day and 0 otherwise (for every node $s_i \in S$ and $q = 1$ or $2$).

The SDB-TSP is as follows.

$$\text{Min} \sum_{(i,j) \in E} \sum_{q=1}^{2} c_{ij} x_{ijq} \quad (1)$$

s.t.
The single-double traveling salesman problem

\[ \sum_{j \in V} x_{jq} = 1 \quad \forall i \in D \cup \{v_0\}; \forall q \quad (2) \]

\[ \sum_{j \in V} x_{jq} = 1 \quad \forall i \in D \cup \{v_0\}; \forall q \quad (3) \]

\[ y_{ii} + y_{i2} = 1 \quad \forall i \in D \cup \{v_0\}; \forall q \quad (4) \]

\[ \sum_{j \in V} x_{ji} = \sum_{j \in V} x_{ji1} = y_{ii} \quad \forall i \in S \quad (5) \]

\[ \sum_{j \in V} x_{ij2} = \sum_{j \in V} x_{ij} = y_{i2} \quad \forall i \in S \quad (6) \]

\[ \sum_{i,j,z} x_{ijq} \leq |Z| - 1 \quad Z \subseteq S \cup D; \forall q \quad (7) \]

\[ \sum_{i \in S} y_{iq} \leq g \ast h \quad \forall q \quad (8) \]

\[ y_{iq} \in \{0,1\} \quad \forall i \in S; \forall q \quad (9) \]

\[ x_{ijq} \in \{0,1\} \quad \forall (i,j) \in E; \forall q \quad (10) \]

The objective function minimizes the total costs.

Constraints (2) and (3) impose that the depot and any double node \( i \in D \) is visited only once (there are one inside-edge and one outside-edge), both for odd \( (q = 1) \) or for even \( (q = 2) \) days. This way, the vehicle will visit any node \( i \in D \) and the depot exactly twice in two days.

Constraints (4) guarantee that every single-node \( i \in S \) is visited only once, or on even or on odd days; for next constraints (5) and (6) there are only one outside edge and one inside edge for any node \( i \in S \).

Constraints (7) are classical subtour elimination constraints of the TSP, \( Z \) being a subset of nodes of the graph \( G \) (see [9] and [10]).

Inequalities (8) represent the ‘balance constraints’; they impose an upper bound to the maximum number of nodes (customers) that can be visited by a vehicle in one day.

There are two limit cases. If \( D = \emptyset \), the problem becomes a VRP in which there are two vehicles (and any vehicle is associated to a day); while, if \( S = \emptyset \) (so that every node has to be visited every day) the problem becomes a TSP.

Quite obviously, the SDB-TSP is a NP-hard problem (see [5], [6], [10]).

2. Two heuristic techniques for SDB-TSP.

A feasible solution for the SDB-TSP can be obtained adjusting some techniques well-known in literature, in particular by using insertion procedures.

In order to take advantages of the peculiarities of the SDB-TSP, we propose two heuristic techniques, A1 and A2, which are particularly suitable when \( G \) is a geometric graph or if it satisfies the triangular property.

A1 is a very simple technique which quickly provides a good solution, in particular when single and double-nodes are uniformly distributed on the region. On the other side, in some cases A1 fails in what it never gives a feasible solution!

The second algorithm, A2, particularly refers to a geometric graph, in such a way that there could be the ‘edge crossing’: we mean, “two different edges cross in a graph drawing if their geometric representations intersect” [1].

Both A1 and A2 are building heuristics and require, as prerequisite:
- (A1), a hamiltonian cycle \( CH \), that visits every node which belongs to \( V \) once;
- (A2), the aforesaid tour \( CH \) and also a cycle \( CH_D \), visiting only the double-nodes and the depot.

In an ideal case, \( CH \) and \( CH_D \) would be minimum length tours: in practice, both cycles \( CH \) and \( CH_D \) can be obtained by an appropriate heuristic [10].

In order to describe the algorithms, some more definitions are needed.

Let “single-node path” \( P(u, v) \), SNP for short, be any path in \( CH \) having node set \( \{ u, i_1, i_2, \ldots, i_r, v \} \), and \( r \geq 2 \) edges, in which the two endpoints, \( u \) and \( v \), are double-nodes, while nodes, \( i_1, i_2, \ldots, i_r \), are single ones. Similarly, we shall call “double-node path” \( Q(u, v) \) the shortest path between nodes \( u \) and \( v \) in \( CH_D \) (obviously, all nodes in a \( Q(u, v) \) path are double-nodes).

Single node paths are univocally defined whenever \( G \) contains more than two double nodes: the case with exactly two double nodes will be treated separately.

Let the cardinality of a single (double) node path be the number of nodes in the path and indicate it by \( | P(u, v) | \) (\( | Q(u, v) | \)).

We shall call ‘\( D \)-edge’ every edge in graph \( G \) whose extreme points are both in \( D \). (Note that \( D \)-edges may be not contained in any \( Q(u, v) \)). The \( D \)-edge \( (u, v) \) joining the extreme point of a single node path will be called the chord of the path.

Both the algorithms build the two tours \( T_1^* \) and \( T_2^* \) stepwise, by the construction of two subgraphs of \( G \), say, \( G_1 \) and \( G_2 \).

In the first step, both \( G_1 \) and \( G_2 \) contain only double-nodes and the depot, but no edges. They become two tours by the addition of single-nodes (that are eventually partitioned between \( G_1 \) and \( G_2 \)) and of edges.

As we can see later, the A1 algorithm doesn’t necessarily build a feasible solution (while this always happens for A2).

The two algorithms A1 and A2 follow (see the appendix for an example).

**A1 algorithm**

*(Remember: a (sub)optimal hamiltonian circuit in \( G \), \( CH \), is given.)*

Step 0. Let \( G_1 = G_2 = (D \cup \{ v_0 \}, \emptyset) \). Go to step 1.

Step 1. If \( D = \{ u \} \), and \( (u, v_0) \subset CH \), then \( T_1^* = CH \) and \( T_2^* = \{ (u, v_0), (v_0, u) \} \) (fig.1). STOP. If \( D = \{ u \} \) where \( u \) and \( v_0 \) are not adjacent in \( CH \) (this way there are two SNPs from \( u \) to \( v_0 \), say \( P_{[u,v_0]} \) and \( P_{[v_0,u]} \)), then insert in \( G_1 \) the path \( P_{[u,v_0]} \) and insert in \( G_2 \) the path \( P_{[v_0,u]} \); insert both in \( G_1 \) and in \( G_2 \) the edge \( (u, v_0) \). Let \( T_1^* = G_1 \) and \( T_2^* = G_2 \) (fig.2). STOP. If \( | D | > 1 \), insert every \( D \)-edge joining double-nodes of \( CH \), if any, both into \( G_1 \) and into \( G_2 \). Go to step 2.
Step 2. Find a partition of the set of all the SNPs in two subsets (call them SNP₁, SNP₂) in such a way that the difference between the total number of single nodes in the paths in SNP₁ and SNP₂ be as less as possible using some heuristics (see, for examples, the hints to solve the PARTITION problem given in [8] and [13]). Go to step 3.

Step 3. Insert all the edges and also single-nodes of the paths in SNP₁ into G₁ and insert their chords into G₂; similarly, insert edges and nodes of the paths contained in SNP₂ into G₂ and their chords into G₁. Go to step 4.

Step 4. If \( k₁ + h ≤ g^* + 1 \) and \( k₂ + h ≤ g^* + 1 \), i.e. if the balancing constraints are satisfied, we have found a feasible solution. Set T₁* = G₁ and T₂* = G₂. STOP.
Otherwise, the solution is not feasible: STOP.

In Step 2 a solution to a SUBSET-SUM problem or, equivalently, to a PARTITION problem, is required. It is well known in literature that both this problems are NP-complete and that “there is a pseudopolynomial algorithm for SUBSET-SUM” (see [8] and [13] for a proof). This way, the overall complexity of the proposed algorithm heavily depends on the techniques we use in order to build the tour CH and to solve the partition subproblem. Excluding these issues, the complexity is clearly polynomial.

In our experience, the use in step 2 of simple heuristics, (such as: insert alternately paths in SNP₁ and paths in SNP₂, sorted in decreasing number of nodes) gives a feasible solution (if it is attainable with simple insertions of SNPs!). This way, the algorithm A₁ is quite easy to implement because it inserts each single-node path directly into G₁ or G₂ (these paths are not split in sub-paths!), but A₁ doesn’t guarantee a feasible solution. On the other hand, A₂ aims to build the tours breaking the single-node paths also, if necessary, before they are inserted.

As we can see from some suitable examples (see the appendix), A₂ takes account also of the fact that the visit order of some double-nodes in the minimum length cycle CHᵟ (that visits only double-nodes) can be different from the visit order in CH. A₂ tries to take advantage from these peculiarities. As we can see later, the heuristic gives interesting results in the tested problems.

Keeping in mind the previous notations, A₂ uses the following procedure:
- in the first step, the depot and all the other double-nodes are inserted both in subgraphs G₁ and G₂;
  - D-edges between double-nodes, which are contained as well in CH as in CHᵟ, are also inserted both in G₁ and in G₂;
  - single-node paths \( P(u, v) \) which do not cross any \( Q(u, v) \) are inserted in G₁ or in G₂;
  - single-node paths \( P(u, v) \) which cross some edges of \( Q(u, v) \) are split into sub-paths: such sub-paths are then inserted in G₁ or in G₂.

After accomplishing these steps, the two subgraphs G₁ and G₂ both consist of some (not necessarily connected) paths: they become two tours linking their endpoints pair-wise in each subgraph and in such a way that there are no subtours.

Then A₂ checks if the balancing constraints (8) are satisfied: if not, there could be some re-allocations of nodes. We want to underline also that, in absence of the balancing constraints, the couple (CH, CHᵟ) is a feasible solution to the problem, but it is not necessarily the optimal solution, as we can see from some suitable examples.
**A2 algorithm**

(Remember: two (sub)optimal hamiltonian circuits in $G$, $CH$ and $CH_D$, are given.)

Step 0. Let $G_1 = G_2 = (D \cup \{v_0\}, \emptyset)$. Order SNPs $\subseteq CH$ by decreasing cardinality of the paths. Go to step 1.

Step 1. Insert all $D$-edges $(u, v) \in CH \cap CH_D$ both into $G_1$ and into $G_2$. Go to step 2.

Step 2. Consider the not yet visited maximum cardinality path $P(u, v) \subseteq CH$. If there are none, go to step 8.

Consider also the double-node path $Q(u, v) \subset CH_D$. Test if $P(u, v)$ crosses $Q(u, v)$ using a crossing-edge test: if not, go to step 3. Otherwise, go to step 6.

Step 3. If any edge $(i, j)$ in $Q(u, v)$ has been already included in $G_1$ or in $G_2$, go to step 4; otherwise, go to step 5.

Step 4. Let $G_i$ be the subgraph which contains (some) edges of $Q(u, v)$; if any edge $(i, j) \in Q(u, v)$ belongs both to $G_1$ and to $G_2$, let $G_i$ be the subgraph for which $|G_i|$ is a maximum. Insert the remaining (if any) edges of $Q(u, v)$ into $G_i$. Insert nodes and edges of $P(u, v)$ into the other subgraph. Go back to step 2.

Step 5. Insert the path $P(u, v)$ into $G_1$ (or $G_2$) and the path $Q(u, v)$ into $G_2$ (or $G_1$) in such a way to insert the maximum cardinality path in the minimum cardinality subgraph and vice-versa. Go back to step 2.

Step 6. Delete out of $P(u, v)$ every edge $(i_j, i_{j+1})$ which crosses the path $Q(u, v)$ and, vice-versa, delete out of $Q(u, v)$ every edge which crosses the path $P(u, v)$.

Let $P_1, P_2, ... P_m$ be the disjoint sub-paths $\subset P(u, v)$ (eventually consisting of only one node) in the sequence in which they are visited in $P(u, v)$; insert single-nodes and paths $P_i$ with odd index into $G_1$ and the ones with even index into $G_2$. Go to step 7.

Step 7. Add the missing edges in order to obtain a path in every subgraph $G_1$ and $G_2$ from double-node $u$ to double-node $v$ of the previous step. Go back to step 2.

Step 8. If both $G_1$ and $G_2$ are tours, let $G_1 = T_1$ and $G_2 = T_2$, then go to step 9. Otherwise, apply to every subgraph that is not a tour the SUBROUTINE EDGE. Go to step 9.

**SUBROUTINE EDGE.**

If $G_i (i = 1, 2)$ is a path from $u$ to $v$, add the missing edge (chord) $(u, v)$ in order to get a tour $T_i$.

If $G_i (i = 1, 2)$ is not connected, consider in $G_i$ the set of double-nodes $D_i \subseteq D$ which have degree equal to one ( $|D_i|$ is an even number and $|D_i| \geq 4$ ). Link pair-wise the nodes in $D_i$ using the nearest-neighbour procedure until $G_i$ is a tour $T_i$.

Step 9. (Remember: $k_1$ denotes the number of single-nodes in $G_1$ and similarly $k_2$ in $G_2$). If $k_1 + h \leq g_* + 1$ and $k_2 + h \leq g_* + 1$, i.e. if the balancing constraints are satisfied, STOP. If not, go to step 10.
Step 10. Suppose, without loss of generality, that T1 is the tour which contains the greater number of nodes and let \( \Delta = k_1 + h - g^* \). Find a single-node path \( P(u, v) \) in T1 for which \(|P| - 2 - \Delta|\) is a minimum and such that the chord \((u, v) \subset T2\). If there is none, go to step 12. Otherwise, swap \( P(u, v) \) with the chord \( Q(u, v) \) between T1 and T2. If \( k_1 + h \leq g^* + 1 \) and \( k_2 + h \leq g^* + 1 \), go to step 11; if once more \( k_1 + h > g^* + 1 \), repeat step 10.

Step 11. If in any tour there is some edge \((i, j)\) crossing another edge \((u, v)\), apply the 2-opt improvement [10]; repeat for all edges. STOP.

Step 12. For every single-node \( s \) in T1 compute the insertion-cost \( r(s) \) given by

\[
r(s) = \min_{(i, j)} c_{is} + c_{sj} - c_{ij},
\]

where the minimum has to be computed with respect to all the edges \((i, j)\) of T2. Go to step 13.

Step 13. Find the single-node \( s^* \) with the minimum insertion-cost. Insert the node \( s^* \) and the edges \((i, s^*)\) and \((s^*, j)\) into T2 deleting the edge \((i, j)\); delete the node \( s^* \) and the two edges of the tour which are linked to it out of T1 and join nodes of degree one by an edge. If \( k_1 + h \leq g^* + 1 \), STOP; otherwise, if once more \( k_1 + h > g^* + 1 \), repeat step 13.

The algorithm A2, as a preliminary step, requires to solve two TSP problems (in order to get both CH and CHD). Excluding from consideration this step, the complexity depends on [9]:

- the complexity of “crossing-edge test” (step 2), i.e. \( O(n^2) \);
- the complexity of the nearest-neighbour procedure (step 8), i.e. \( O(n^2) \);
- the complexity of the 2-opt procedure (step 11), i.e. \( O(n^2) \);
- the complexity of the insertion cost of a node (step 12 and 13), i.e. \( O(n^2) \).

Observations. From a practical point of view, the best solution cost given by the first algorithm is less than twice the cost of CH. A1 visits double-nodes, by alternating them with single-nodes in the two subtours, but always in the same order.

Vice-versa, as one can see from suitable examples (remember: G is an undirected graph!) the optimal solution requires that now and then the order in which double-nodes are visited could be changed by swapping paths from the first to the second subtour: we take account of this feature in the second heuristic A2. Finally, even if we did not find peculiar difficulties in our computational experiences, the computational cost of step 12 could be high (in our examples, step 12 is rarely necessary).

In the appendix we give a suitable enforcing example of the modus operandi of the two heuristics.

5. Concluding remarks.

In this paper we have introduced, formulated and solved the Single-Double Balanced TSP. Two heuristic algorithms are proposed for its solution.

Good results are given by heuristics, for which we have proposed quick methods for building a solution and also techniques for improving it. The performance of the heuristics depends above all on the quality of the initial solution and on the chosen improving methods. Different improving methods lead to algorithms more or less efficacious according to objectives; this way the computational cost can change very deeply. In some tested instances, for which we are able to find the optimal solution using a branch and bound approach, A2 algorithm gives solutions really near to the optimal one,
due to the fact that it is able to change the sequence of the visits of some double-nodes with respect to the order in CHD.

An other interesting generalization of the problem, that can have also an interesting practical application, is concerned with the utilization of more vehicles and of different cost functions linked to the visit of nodes.

References


In this appendix we give an example of the two heuristics. Let us consider the graph in fig. 3, in which double-nodes and the depot are black and single-nodes are white and in which distances are Euclidean (at the end of this appendix we find all the figures).

In this example we use software available in [http://mathsrv.kueichstaett.de/MGF/homes/grothmann/java/tsp.html](http://mathsrv.kueichstaett.de/MGF/homes/grothmann/java/tsp.html) in order to generate nodes in a random way and to get CH and CHD.

We generate 15 nodes: 5 of them are double-nodes (including the depot), i.e. the set $D \cup \{v_0\} = \{a, f, i, l, n\}$ (black ones); the remaining ones are single-nodes and form the set $S = \{b, c, d, e, g, h, m, o, p, q\}$ (white ones).

The two starting tours are (fig. 4):

CH = \{(a,b),(b,c),(c,d),(d,e),(e,f),(f,g),(g,h),(h,i),(i,l),(l,m),(m,n),(n,o),(o,p),(p,q),(q,a)\};

CHD = \{(a,n),(n,l),(l,f),(f,i),(i,a)\}.

**A1 algorithm**

Step 0. $G_1 = G_2 = \{\{a, f, i, l, n\}, \emptyset\}$ (fig. 5). Go to step 1.

Step 1. Since $|D| = 5 > 1$, insert in both subgraphs the $D$-edge $(i, l)$, obtaining $G_1 = G_2 = \{\{a, f, i, l, n\}, (i, l)\}$. Go to step 2.

Step 2. Starting from the double-node $a$, we individuate four single-node paths:

$P(a,f) = \{(a,b),(b,c),(c,d),(d,e),(e,f)\};$

$P(f,i) = \{(f,g),(g,h),(h,i)\};$

$P(l,n) = \{(l,m),(m,n)\};$

$P(n,a) = \{(n,o),(o,p),(p,q),(q,a)\}.$

Let SNP$_1 = P(a,f) \cup P(l,n)$ and SNP$_2 = P(f,i) \cup P(n,a)$. Go to step 3.

Step 3. We insert SNP$_1$ into $G_1$ and SNP$_2$ into $G_2$. Insert the chords $(f, i)$ and $(n, a)$ into $G_1$; and the chords $(a, f)$ and $(l, n)$ into $G_2$. This way, we obtain

$G_1 = \{\{a, b, c, d, e, f, i, l, m, n\}, \{a,b),(b,c),(c,d),(d,e),(e,f),(f,i),(i,l),(l,m),(m,n),(n,a)\};$

$G_2 = \{\{a, f, g, h, i, l, n, o, p, q\}, \{a,f),(f,g),(g,h),(h,i),(i,l),(l,n),(n,o),(o,p),(p,q),(q,a)\}\}$. Go to step 4.

Step 4. Since $k_1 + h = k_2 + h = 10$, the two tours are a feasible solution. This way, $T_1 = \{\{a, b, c, d, e, f, i, l, m, n\}, \{a,b),(b,c),(c,d),(d,e),(e,f),(f,i),(i,l),(l,m),(m,n),(n,a)\};$

$T_2 = \{\{a, f, g, h, i, l, n, o, p, q\}, \{a,f),(f,g),(g,h),(h,i),(i,l),(l,n),(n,o),(o,p),(p,q),(q,a)\}\}$. STOP.

In this example the heuristic A1 gives a feasible solution in a limited number of steps, but not the optimal solution, as we can see from the heuristic A2.

**A2 algorithm**

Step 0. $G_1 = G_2 = \{\{a, f, i, l, n\}; \emptyset\}$ (fig. 5). The ordered SNPs are

$P(a,f) = \{(a,b),(b,c),(c,d),(d,e),(e,f)\};$

$P(n,a) = \{(n,o),(o,p),(p,q),(q,a)\};$

$P(f,i) = \{(f,g),(g,h),(h,i)\};$

$P(l,n) = \{(l,m),(m,n)\}$. Go to step 1.
Step 1.  \(\text{CH} \cap \text{CH}_D = \emptyset\). Then \(G_1 = G_2 = \{(a, f, i, l, n); \emptyset\}\) (fig.5). Go to step 2.

Step 2.  Consider the single-node path \(P(a, f)\) in \(\text{CH}\) and \(Q(a, f)\) in \(\text{CH}_D\); since \(P(a, f)\) doesn’t cross \(Q(a, f)\), go to step 3.

Step 3.  None edge of \(Q(a, f)\) has been already included in \(G_1\) or \(G_2\). Go to step 5.

Step 5.  Insert \(P(a, f)\) in \(G_1\) and \(Q(a, f)\) in \(G_2\), obtaining
\[
G_1 = \{(a, b, c, d, e, f, i, l, n); \{(a,b) (b,c) (c,d) (d,e) (e,f)\}\};
G_2 = \{(a, f, i, l, n); \{(a,n) (n,l) (l,f)\}\} \text{ (fig.6). Return to step 2.}

Step 2.  Consider \(P(a, n)\) in \(\text{CH}\) which doesn’t cross \(Q(a, n)\) in \(\text{CH}_D\). Go to step 3.

Step 3.  The edge \((a, n)\) is just inside \(G_2\). Go to step 4.

Step 4.  Let \(G_i = G_2\). Insert \(P(a, n)\) into \(G_1\),
\[
G_1 = \{a, b, c, d, e, f, i, l, n, o, p, q\}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f,g) (n, o), (o, p), (p, q), (q, a)\};
G_2 = \{a, f, i, l, n\}; \{(a,n) (n,l) (l,f)\}\} \text{ (fig.7). Return to step 2.}

Step 2.  Consider the single-node path \(P(f, i)\) in \(\text{CH}\) that crosses the edge \((f, i)\) in \(\text{CH}_D\). Go to step 6.

Step 6.  Delete edges \((g, h)\) and \((f, i)\). Let \(P_1 = (f, g)\) and insert it into \(G_1\) and \(P_2 = (h, i)\) and insert it into \(G_2\). We obtain
\[
G_1 = \{a, b, c, d, e, f, g, i, l, n, o, p, q\}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f,g) (n, o), (o, p), (p, q), (q, a)\};
G_2 = \{a, f, i, l, n, h, i\}; \{(a,n) (n,l) (l,f) (h,i)\}\}. Go to step 7.

Step 7.  Insert the two edges \((g, i)\) into \(G_1\) and \((f, h)\) into \(G_2\), obtaining
\[
G_1 = \{a, b, c, d, e, f, g, i, l, n, o, p, q\}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f,g) (g,i) (n, o), (o, p), (p, q), (q, a)\};
G_2 = \{a, f, i, l, n, h, i\}; \{(a,n) (n,l) (l,f) (h,i)\}\} \text{ (fig.8). Return to step 2.}

Step 2.  Consider \(P(l, n)\) in \(\text{CH}\): it doesn’t cross \(Q(l, n)\) in \(\text{CH}_D\). Go to step 3.

Step 3.  The \(D\)-edge \((l, n)\) is just inside \(G_2\). Go to step 4.

Step 4.  Let \(G_i = G_2\). Insert \(P(l, n)\) into \(G_1\),
\[
G_1 = \{a, b, c, d, e, f, g, i, l, m, n, o, p, q\}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f,g) (g,i) (l, m) (m, n) (n, o), (o, p), (p, q), (q, a)\};
G_2 = \{a, f, i, l, n, h, i\}; \{(a,n) (n,l) (l,f) (h,i)\}\} \text{ (fig.9). Return to step 2.}

Step 2.  Since there is none SNP, go to step 8.

Step 8.  Apply the SUBROUTINE EDGE.
Since \(G_1\) is a connected path, add the edge \((i, l)\) in \(G_1\); since \(G_2\) is also a connected path, add the edge \((i, a)\) in \(G_2\), obtaining the two tours:
\[
T_1 = \{a, b, c, d, e, f, g, i, l, m, n, o, p, q\}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f,g) (g,i) (i,l) (l, m) (m, n) (n, o), (o, p), (p, q), (q, a)\};
\]
The single-double traveling salesman problem

\[ T_2 = \{ \{ a, f, l, n, h, i \}; \{(a,n) (l,f) (f,h) (h,i) (i,a)\} \} \] (fig.10). Go to step 9.

Step 9. The found solution is feasible if \[ g^* \geq 13 \] because \[ |T_1| = 14 \]. In this case, STOP. For this example, assume \[ g^* = 10 \]. This way, go to step 10.

Step 10. \[ |T_1| > |T_2| \] (\( \Delta = 4 \)). Don’t swap \( P(a, f) \) because \( Q(a, f) \) is not a chord. Swap \( P(a, n) \) with the chord \( (a, n) \):

\[ T_1 = \{ \{ a, b, c, d, e, f, g, i, l, m, n \}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f, g) (g, i) (i, l) (l, m) (m,n) (n, a)\} \} \]

\[ T_2 = \{ \{ a, f, i, l, n, h, o, p, q \}; \{(n, o) (o, p) (p, q) (q, a) (n, l) (l,f) (f, h) (h, i) (i, a)\} \}. \]

Since \( k_1 + h = 11 > 10 \), repeat step 10.

Step 10. Swap \( P(l, n) \) with the chord \( Q(l, n) \). We obtain

\[ T_1 = \{ \{ a, b, c, d, e, f, g, i, l, n \}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f, g) (g, i) (i, l) (l, n) \} \} \]

\[ T_2 = \{ \{ a, f, i, l, m, n, h, o, p, q \}; \{(n, o) (o, p) (p, q) (q, a) (n, m) (m, l) (l,f) (f, h) (h, i) (i, a)\} \} \] (fig.11). Since \( k_1 + h = k_2 + h = 10 \), go to step 11.

Step 11. \( (q, p) \) crosses \( (a, i) \) in \( G_2 \), then delete \( (q, p) \) and \( (a, i) \) and insert \( (a, p) \) and \( (i, q) \) obtaining

\[ T_1^* = \{ \{ a, b, c, d, e, f, g, i, l, n \}; \{(a,b) (b,c) (c,d) (d,e) (e,f) (f, g) (g, i) (i, l) (l, n) (n, a)\} \} \]

\[ T_2^* = \{ \{ a, f, i, l, m, n, h, o, p, q \}; \{(n, o) (o, p) (p, a) (q, a) (n, m) (m, l) (l,f) (f, h) (h, i) (i, q)\} \} \] (fig. 12). STOP.

For this example, the algorithm A2 finds a feasible solution, i.e. a solution better than the one found by A1 algorithm.

---

**Fig.3**

**Fig.4**
Fig. 5  $G_1 = G_2$

Fig. 6

Fig. 7
The single-double traveling salesman problem

Fig. 8

G1

G2

Fig. 9

G1

G2

Fig. 10

T1

T2