Math. Prog. Comp. (2011) 3:59–78
DOI 10.1007/s12532-010-0021-5

FULL LENGTH PAPER

Computing compatible tours for the symmetric traveling salesman problem

Matteo Fortini · Adam N. Letchford · Andrea Lodi · Klaus M. Wenger

Received: 26 October 2009 / Accepted: 29 November 2010 / Published online: 29 December 2010

Abstract We consider the following natural heuristic for the Symmetric Traveling Salesman Problem: solve the subtour relaxation, yielding a solution x^*, and then find the best tour $\bar{x}$ that is *compatible* with x^*, where *compatible* means that every subtour elimination constraint that is satisfied at equality at x^* is also satisfied at equality at $\bar{x}$. We prove that finding the best compatible tour is $\mathcal{NP}$-hard and show that the tour can have a cost approaching $5/3$ that of the optimal tour. We then describe a branch-and-cut algorithm for computing the best compatible tour, and present extensive computational results for TSPLIB instances. It turns out that, in practice, the tour is usually of very good quality. Moreover, the computational effort for computing the compatible tour is considerably smaller than that of solving the full problem with the best available software, i.e., `Concorde`.

Keywords Traveling salesman problem · Heuristics · Branch-and-cut

Mathematics Subject Classification (2000) 90C27 · 90C57 · 90C59

M. Fortini · A. Lodi
D.E.I.S., University of Bologna, Bologna, Italy
e-mail: matteo.fortini@gmail.com

A. Lodi
e-mail: andrea.lodi@unibo.it

A. N. Letchford (✉)
Department of Management Science,
Lancaster University, Lancaster, United Kingdom
e-mail: A.N.Letchford@lancaster.ac.uk

K. M. Wenger
Institute of Computer Science, University of Heidelberg,
Heidelberg, Germany
e-mail: klaus_wenger@yahoo.com

1 Introduction

The *symmetric traveling salesman problem*, or STSP, is the problem of finding a minimum weight Hamiltonian circuit in an edge-weighted graph. The STSP is a fundamental problem in combinatorial optimisation, and has received a huge amount of attention in the literature (see the books Lawler et al. [14], Gutin and Punnen [11] and Applegate et al. [1]).

Although the STSP is $\mathcal{NP}$-hard, large instances can often be solved to proven optimality via *branch-and-cut* (e.g., Padberg and Rinaldi [17] or Applegate et al. [1]). Branch-and-cut algorithms for the STSP are based on the fact that the STSP can be formulated as the following 0-1 Linear Program (0-1 LP):

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e \\
\text{s.t.} \quad & \sum_{e \in \delta(\{i\})} x_e = 2 \quad (i \in V) && (1) \\
& \sum_{e \in \delta(S)} x_e \geq 2 \quad (S \subset V : 2 \leq |S| \leq |V|/2) && (2) \\
& x_e \in \{0, 1\} \quad (e \in E).
\end{aligned}$$

Here, V is the vertex set, E is the edge-set, c_e denotes the cost of the edge e and x_e is a binary variable, taking the value 1 if and only if e is in the tour. For a given vertex set S, the term $\delta(S)$ denotes the set of edges having exactly one end-vertex in S. The constraints (1) are called *degree equations* and the constraints (2) are called *subtour elimination constraints* (SECs).

The linear programming (LP) relaxation of the above 0-1 LP is sometimes called the *subtour relaxation* of the STSP. The first step in a branch-and-cut algorithm for the STSP is usually to solve the subtour relaxation via a cutting-plane method. The solution of the subtour relaxation yields a lower bound on the cost of the optimal tour. In practice, this lower bound is rather strong (see, e.g., [1]). Indeed, it has been conjectured (see Goemans [10]) that the worst-case ratio between the optimum and the lower bound is never more than $4/3$, when the STSP instance is metric (i.e., has edge costs satisfying the triangle inequality).

Now, let x^* be a basic optimal solution to the subtour relaxation. Although x^* is typically fractional, it seems reasonable to suppose that x^* will contain some useful information that could be exploited. This is the idea underlying this paper, in which we define a *compatible tour* as a vector $\bar{x}$ that is *compatible* with x^* in the sense that every SEC that is satisfied at equality at x^* is also satisfied at equality at $\bar{x}$. (The reverse need not be true.) Clearly, such a compatible tour is a heuristic solution of the STSP.

The paper is structured as follows. In Sect. 2, we recall some known results on the structure of sets that satisfy SECs at equality, which we call *tight sets*. In Sect. 3, we prove three negative theoretical results, one of which is that the problem of finding a best compatible tour is strongly $\mathcal{NP}$-hard, even when the instance is metric. In Sect. 4, we describe a branch-and-cut algorithm for finding a best compatible tour. Extensive computational results are given in Sect. 5, showing that the heuristic solutions obtained are typically of good quality, and that finding the compatible tour is much faster than solving the STSP. Finally, some concluding remarks are given in Sect. 6, in which we discuss the potential application of the compatible tour idea to cutting plane generation and branching schemes for the STSP.

We close this introduction with two important remarks. First, there may exist more than one basic optimal solution to the subtour relaxation. In this case, we assume that the compatible tour heuristic simply chooses one such solution arbitrarily. We will see that the choice of the solution can affect the performance of the heuristic, both in theory (Subsect. 3.2) and practice (Sect. 5). Second, one can easily show that the number of compatible tours can grow exponentially with the number of vertices. Thus, the heuristic selects a best tour from an exponentially-large collection of tours. Some other heuristics of this type have been proposed recently (Deineko and Tiskin [6]; Letchford and Pearson [15]).

2 The structure of tight sets

In this section, we give some basic definitions and notation, and recall some known facts about the structure of tight sets.

From now on, we let n denote the number of vertices, and assume without loss of generality that the graph $G = (V, E)$ is complete. The set of feasible solutions to the subtour relaxation, i.e., the set

$$\left\{ x \in [0,1]^{\binom{n}{2}} : (1), (2) \quad \text{hold} \right\},$$

is called the *subtour elimination polytope* and denoted by SEP(n) (Boyd and Pulleyblank [3]). Using a modern LP-based cutting-plane algorithm, one can compute a basic optimal solution to the subtour relaxation quickly [1,17]. So, we can assume that x^* is an extreme point of SEP(n).

Now, define the edge set $E^* = \{e \in E : x^*_e > 0\}$. The graph $G^* = (V, E^*)$ is called the *support graph* of x^*. For each edge $e \in E^*$, the value x^*_e is called the *weight* of the edge.

We now formally define *tight sets* and *compatible tours*:

Definition A vertex set $S \subset V$ is said to be *tight* (at x^*) if $x^*(\delta(S)) = 2$.

Each tight set S corresponds to a cut $\delta(S)$ of weight 2 in the weighted support graph. Since x^* satisfies the degree equations and SECs by assumption, each such cut is a minimum weight cut in G^*.

Definition We say that a tour is *compatible* with x^* if the associated incidence vector ($\bar{x}$, say) has the following property: every vertex set which is tight at x^* is also tight at $\bar{x}$.

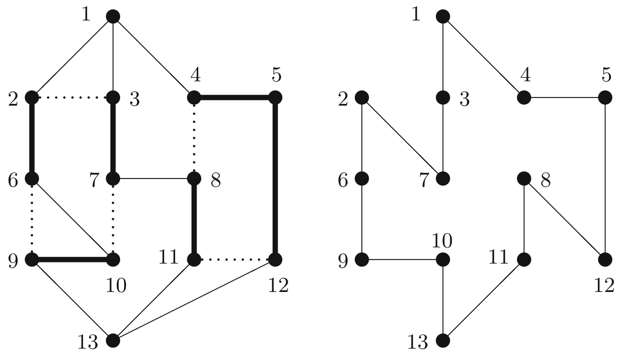

Fig. 1 An extreme point x^* of SEP(13) and a compatible tour $\bar{x}$

Fig. 2 Cactus corresponding to the extreme point shown on the *left* of Fig. 1

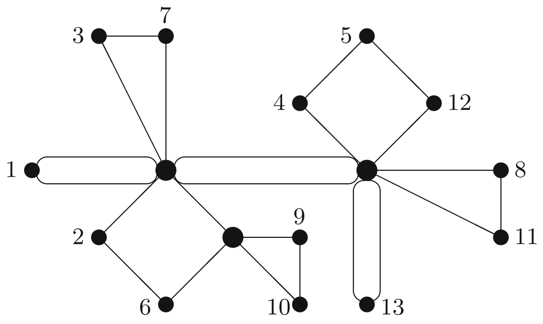

Figure 1 illustrates this concept. An extreme point x^* of SEP(13) is shown on the left. The thick, thin and dotted lines represent variables with value $1, 2/3$ and $1/3$, respectively. The sets {2, 6}, {6, 9, 10} and {4, 5, 8, 11, 12, 13}, for example, are tight. A tour $\bar{x}$ compatible with the extreme point is shown on the right.

Dinitz et al. [7] presented a compact (linear-space) data structure for representing and storing the minimum cuts of any undirected graph with non-negative edge weights: the so-called *cactus* of the graph. A cactus is a connected, undirected and unweighted graph in which every edge appears in exactly one circuit. Pairs of parallel edges are permitted, in which case they form a 'degenerate' circuit. Each vertex of the original graph is assigned to a node of the cactus, but the cactus may have additional nodes with no vertex of the graph assigned to them. Each minimum cut in the cactus corresponds to a minimum weight cut in the original graph. Figure 2 shows a cactus corresponding to the extreme point x^* shown in Fig. 1.

Observe that, if we construct a cactus corresponding to an STSP support graph G^*, then there is a one-to-one correspondence between the compatible tours and the possible *Eulerian traversals* of the cactus (i.e., traversals of the cactus that traverse each edge exactly once). This means that, in general, the number of compatible tours can grow exponentially with n.

One more concept from [7] that we will find useful is that of a *circular partition* (called a *necklace* in Applegate et al. [1]):

Definition 1 A *circular partition* is a partition of V into subsets $S_1, \dots, S_q$, with $q \geq 3$, such that the union $\bigcup_{k=i}^{k=i+j} S_k$ is tight for any $1 \leq i \leq q$ and any $0 \leq j < q-1$ (indices taken modulo q), and no finer such partition exists. The individual S_k are the *beads* of the partition.

For example, for the extreme point shown on the left of Fig. 1, one circular partition has the beads {2}, {6}, {9, 10} and $V\backslash\{2, 6, 9, 10\}$, and another has the beads {4}, {5}, {12} and $V\backslash\{4, 5, 12\}$.

De Vitis [19] showed that one can always construct a cactus (called the *canonical* cactus) such that there is a one-to-one correspondence between the circular partitions in the original graph and the non-degenerate circuits in the cactus. Such a cactus can be computed in polynomial time via a variety of algorithms [8,16,20,21].

Note that there can exist tight sets that do not belong to any circular partition. For the point shown in Fig. 1, the sets {1} and {4, 5, 8, 11, 12, 13} are examples. Such sets correspond to degenerate circuits in the cactus.

It is known that the cactus of an n-vertex graph contains $\mathcal{O}(n)$ nodes and edges. This implies that the number of beads, and therefore also circular partitions, is $\mathcal{O}(n)$. It also implies that the total number of tight sets is $\mathcal{O}(n^2)$.

Finally, we remark that Applegate et al. [1] showed that one can use the *PQ-tree* of Booth and Lueker [2] in place of the cactus. The Q-nodes of a PQ-tree correspond to circuits in the cactus. The P-nodes correspond to nodes in the cactus whose removal causes the cactus to break apart into four or more connected components. In Fig. 2, there are two such nodes.

In this paper, we use the cactus, rather than the PQ-tree, to represent the minimum cuts of the support graph G^*.

3 Some theoretical results

In this section, we prove three theoretical results which, unfortunately, are all of a negative nature.

3.1 The complexity of finding a best compatible tour

First, we will show that finding a best compatible tour is an $\mathcal{NP}$-hard problem. For this, we need the following well-known result.

Proposition 1 (Garey et al. [9]) *Testing if a cubic (i.e., 3-regular) graph is Hamiltonian is $\mathcal{NP}$-complete in the strong sense.*

We will also need the following lemma.

Lemma 1 *Let $G = (V, E)$ be a cubic Hamiltonian graph and let $C \subset E$ form a Hamiltonian circuit. Let G' be the multigraph obtained by duplicating the edges in $E\backslash C$. Then G' is Eulerian and there exists a traversal of G' in which the edges of C are visited in the same order that they are visited in the Hamiltonian circuit itself.*

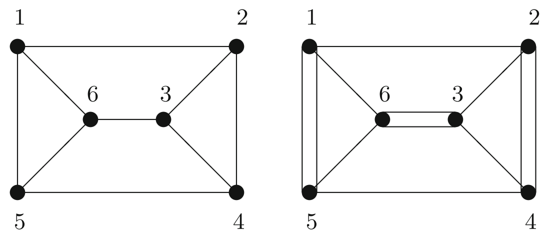

Fig. 3 Cubic Hamiltonian graph G and Eulerian multigraph G'

Proof Since G is cubic and $E\backslash C$ forms a perfect matching, G' is 4-regular. Since G is Hamiltonian, G' is connected. So G', being connected and having even vertex degrees, is Eulerian. We can easily construct the desired traversal of G' by taking the sequence of edges of the Hamiltonian circuit and inserting each pair of parallel edges in an appropriate place in the sequence. Specifically, a pair of parallel edges $\{i, j\}, \{j, i\}$, with $i < j$, can be traversed immediately after vertex i is visited in the Hamiltonian circuit. □

Example The cubic graph G on the left of Fig. 3 is Hamiltonian. A suitable set C consists of the edges $\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}$ and $\{6, 1\}$. The resulting multigraph G' is displayed on the right of the figure. There are three edge pairs that need to be inserted in the sequence. The pair $\{2, 4\}, \{4, 2\}$ is inserted between $\{1, 2\}$ and $\{2, 3\}$, the pair $\{3, 6\}, \{6, 3\}$ is inserted between $\{2, 3\}$ and $\{3, 4\}$, and the pair $\{1, 5\}, \{5, 1\}$ is inserted between $\{6, 1\}$ and $\{1, 2\}$. The resulting traversal of G' is $\{1, 2\}, \{2, 4\}, \{4, 2\}, \{2, 3\}, \{3, 6\}, \{6, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}, \{6, 1\}, \{1, 5\}, \{5, 1\}$.

We are now ready to prove the hardness result.

Theorem 1 *Finding a best compatible tour is $\mathcal{NP}$-hard in the strong sense.*

Proof We reduce the problem of testing if a cubic graph is Hamiltonian to the compatible tour problem. Let $G = (V, E)$ be an arbitrary cubic graph, with n vertices and $3n/2$ edges. Without loss of generality, we can assume that G is biconnected (i.e., contains no cut-vertices) since, if not, it is clearly non-Hamiltonian.

We construct an instance of the STSP on $3n$ vertices as follows. For each edge $\{i, j\}$ in E, we have two vertices, labelled v_{ij} and v_{ji}. For all $\{i, j\}$ in E, we set the cost of the edge connecting v_{ij} and v_{ji} to 0. For any two adjacent edges $\{i, j\}, \{i, k\}$ in E, we set the cost of the edge connecting v_{ij} and v_{ik} to 1. We also set the cost of the edge connecting v_{ij} and v_{ki}, and that of the edge connecting v_{ik} and v_{ji}, to some large positive integer M. Finally, for all remaining edges, say connecting v_{ip} and v_{jq}, we set the cost to M^2.

It is not difficult to show that the unique optimal solution to the subtour relaxation is 1/2-integral, with the following structure. For each edge of zero cost, the corresponding variable takes the value 1. For each edge of cost 1, the corresponding variable takes the value 1/2. The remaining variables take the value 0. Since there are $3n$ edges of cost 1, the total cost of this 1/2-integral solution is $3n/2$.

Figure 4 illustrates the 1/2-integral solution corresponding to the cubic biconnected graph illustrated in Fig. 3. Solid (respectively, dotted) lines represent edges whose variables have value 1 (respectively, 1/2).

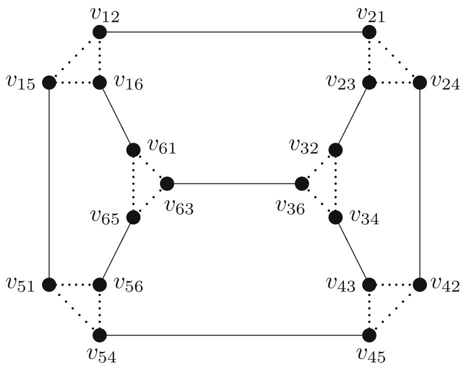

Fig. 4 Extreme point of SEP(18)

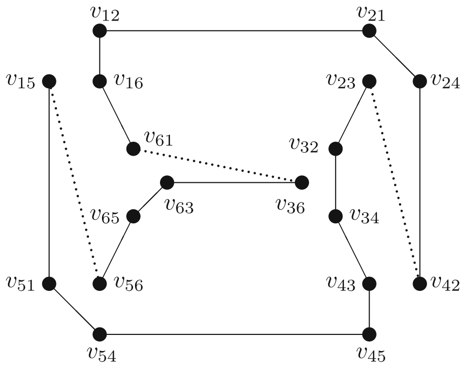

Fig. 5 Compatible tour corresponding to traversal of G'

Note that each of the $3n/2$ edges of zero cost forms a tight set. Any compatible tour must traverse these $3n/2$ edges. Now, note that, since all vertex degrees are odd in a cubic graph, any compatible tour must use at least $n/2$ of the expensive edges (i.e., those of cost M or M^2). In order to minimise costs, a best compatible tour will use, if possible, no edges of cost M^2 and exactly $n/2$ edges of cost M. Such a tour also uses exactly n edges of cost 1, and therefore has a total cost of $n(1 + M/2)$. (Fig. 5 illustrates such a compatible tour for our example. The 3 dotted lines represent the edges of cost M.)

Now we show that, if G is Hamiltonian, then such a compatible tour exists. By Lemma 1, given any Hamiltonian circuit in G there exists a traversal of G' in which the edges are visited in the same order that they are visited in the circuit. We show that such a traversal can be extended to a compatible tour of the desired form for the STSP instance. For any node i of G, let $\{i, j\}$ be the (unique) edge in G which does not appear in the Hamiltonian circuit, and which therefore appears twice in G'. We distinguish two cases:

1. the traversal of G' contains a subsequence of the form k, i, j, i, ℓ;
2. the traveral of G' contains a subsequence of the form k, i, ℓ and another subsequence of the form j, i, j.

In the first case, we include the edges $\{v_{ki}, v_{ik}\}$, $\{v_{ik}, v_{ij}\}$, $\{v_{ij}, v_{ji}\}$ and $\{v_{ji}, v_{i\ell}\}$ in the compatible tour. In the second, we include the edges $\{v_{ki}, v_{ik}\}$ and $\{v_{ik}, v_{i\ell}\}$ in the

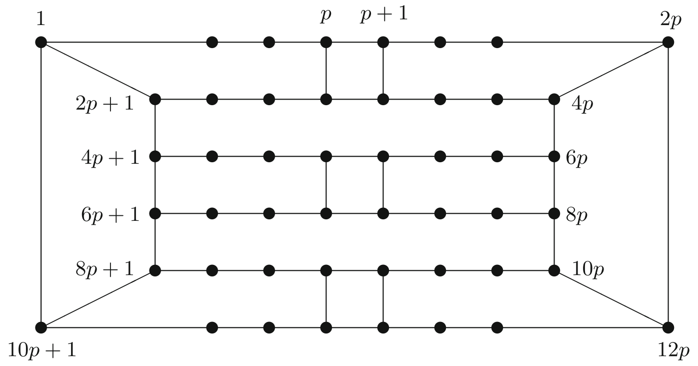

Fig. 6 The graph H_4

compatible tour. Note that the resulting compatible tour contains $n/2$ edges of cost M and n edges of cost 1, as desired.

Finally, we show that, if a compatible tour of the desired form exists, then G is Hamiltonian. To do this, we perform the following two steps. First, we transform the compatible tour into a traversal of G', by shrinking each cluster of three nodes into a single node. Second, we 'short-cut' the traversal of G', by omitting the edge-pairs in the duplicated matching, to yield a Hamiltonian circuit of G. □

Note that the hardness result still holds even when the instance is metric, since any STSP instance can be made metric by adding a large constant to the cost of every edge.

3.2 On the approximation ratio

Now we move on to our second negative result, which is stated in the following theorem.

Theorem 2 *The compatible tour heuristic can return a tour whose cost is arbitrarily close to* $5/3$ *times the cost of the optimal tour, even when the instance is metric and even when the subtour lower bound is equal to the cost of the optimal tour.*

Proof For any integer $p \geq 2$, we construct a graph H_p as follows (see Fig. 6 for an illustration of the graph H_4). The vertex set is $\{1, \ldots, 12p\}$. For $i = 0, \ldots, 5$, and $j = 1, \ldots, 2p - 1$, the edge $\{2ip + j, 2ip + j + 1\}$ is present. For $i = 0, \ldots, 5$, the edges $\{2ip+1, 2(i+1)p+1\}$ and $\{2(i+1)p, 2(i+2)p\}$ are present. For $i = 0, \ldots, 2$, the edges $\{(4i+1)p, (4i+3)p\}$ and $\{(4i+1)p+1, (4i+3)p+1\}$ are present. Finally, the edges $\{1, 10p + 1\}$ and $\{2p, 12p\}$ are present.

Associated with the graph H_p, we define an STSP instance with $12p$ vertices as follows. If the edge $\{i, j\}$ appears in H_p, then the cost of that edge is set to 1 in the STSP instance. Otherwise, the cost is set equal to the number of edges in the shortest path from i to j in H_p. The resulting costs clearly satisfy the triangle inequality.

It is easy to check that several Hamiltonian circuits exist in the graph H_p. One such circuit is shown in Fig. 7 for H_4. Since each edge used in such a tour has a cost of 1,

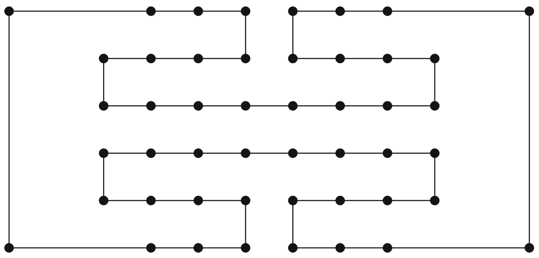
Fig. 7 A Hamiltonian circuit in H_4

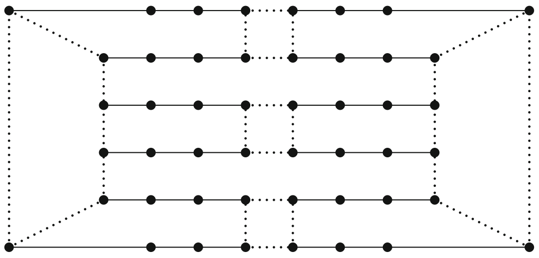
Fig. 8 Optimal fractional vertex of SEP(48)

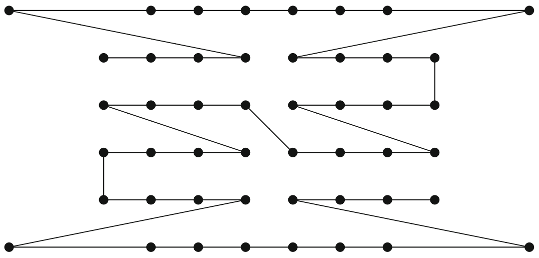
Fig. 9 Compatible tour for H_4 (one edge omitted for clarity)

each of these circuits represents an optimal solution of the STSP instance with a cost of $12p$.

Since every point in SEP($12p$) satisfies $x(E) = 12p$, and every edge has a cost of at least 1, the subtour lower bound is equal to the cost of the optimal tour(s) for this family of instances. That is, any optimal tour is also an optimal solution to the subtour relaxation. Moreover, there exist many alternative optimal solutions to the subtour relaxation that are fractional. One such fractional solution is shown in Fig. 8. As before, solid (respectively, dotted) lines represent edges e with $x^*_e = 1$ (respectively, $x^*_e = 1/2$). There are $12(p-1)$ edges with $x^*_e = 1$ and 24 edges with $x^*_e = 1/2$. Since all of these edges have cost 1, the cost of the fractional solution is also equal to $12p$ as stated.

One can check by brute-force enumeration that one of the best tours compatible with the fractional point of the form displayed in Fig. 8 is of the form displayed in Fig. 9. Such tours use $12p - 8$ edges of cost 1, one edge of cost 2, six edges of cost p, and one edge of cost $2p + 2$ (which is not shown in the figure). Their total cost is therefore $20p - 4$. As p approaches infinity, the ratio of $20p - 4$ to $12p$ approaches $5/3$. ◻

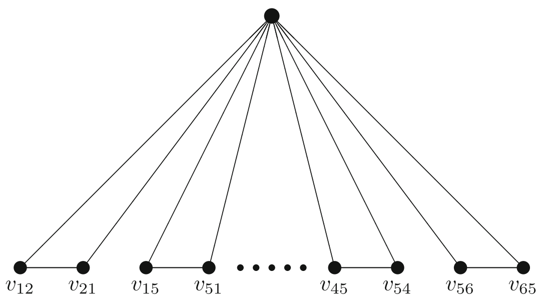

Fig. 10 Cactus corresponding to the fractional point in Fig. 4

3.3 On cut-nodes in the cactus

Burkhard et al. [4] presented a dynamic programming algorithm for finding a best tour compatible with a given PQ-tree, that runs in polynomial time if the degree of each P-node in the PQ-tree is bounded by a constant. In our terms, this means that the algorithm runs in polynomial time if each cut-node in the cactus tree has bounded degree. Unfortunately, we have the following negative result:

Theorem 3 *Cacti associated with extreme points of the subtour polytope can contain cut-nodes of arbitrarily large degree.*

Proof Consider again the proof of Theorem 1 and assume that the biconnected cubic graph G is also 3-edge-connected. Then, in the fractional point for the associated STSP instance, all of the tight sets have cardinality 2. Indeed, there are precisely $|E|$ such tight sets, one for each edge of E. (Since G is 3-edge-connected, all other vertex sets $S \subset V'$ satisfy $x^*(\delta(S)) \geq 3$.) Therefore, the cactus contains a single cut-node, of degree $|E|$, and $|E|$ non-degenerate circuits (see Fig. 10). Since biconnected, cubic, 3-edge-connected graphs can be arbitrarily large, the result follows. □

Fortunately, we will see in the next section that a best compatible tour can often be found quickly via branch-and-cut.

4 An algorithm for finding a best compatible tour

In this section, we describe an algorithm for computing a basic optimal solution x^* to the subtour relaxation, and then finding a best tour $\bar{x}$ that is compatible with it. The algorithm has three main phases: solving the subtour relaxation, extracting tight sets, and performing branch-and-cut. These phases are described in the following three subsections.

4.1 Phase 1: solving the subtour relaxation

First, we have to solve the subtour relaxation. This is done with the following algorithm:

1. Compute an initial feasible tour T, using a greedy heuristic.

2. Construct an initial edge set E', which includes all of the edges in T, plus the p_1 shortest edges incident on each vertex, where p_1 is a parameter.
3. Construct an initial LP relaxation, containing one variable for each edge in E', one degree equation (1) for each vertex, and trivial lower and upper bounds for each variable.
4. Solve the initial LP relaxation via the primal simplex method. (Note that the relaxation is feasible by construction.)
5. Call SEC separation to search for violated SECs. If no violated SECs are found, go to step 7.
6. Add the violated SECs to the LP, re-optimise via dual simplex, and return to step 5.
7. Check for variables with negative reduced cost. If none are found, stop.
8. Add the variables with negative reduced cost to the LP, re-optimise via primal simplex, and return to step 5.

The routines for the greedy heuristic and for the separation of SECs are taken from `Concorde`, a well-known free software package for solving the STSP [1,5]. For solving LPs, we used the `primopt` and `dualopt` routines from the callable library version of `CPLEX` 10.0.

To improve the performance of this algorithm, we 'clean' the LP each time it is re-optimised. Specifically, we delete variables that have been non-basic for p_2 iterations, and delete SECs that have been non-binding for p_2 iterations, where p_2 is another parameter.

4.2 Phase 2: extracting tight sets

Solving the subtour relaxation yields a (typically fractional) point $x^* \in \text{SEP}(n)$. The next step is to compute the cactus representation of the minimum cuts of the support graph G^*. We use the cactus construction algorithm of Wenger [20,21]. It runs in $\mathcal{O}(nm \log n \log(n^2/m))$ time, where $m = |E^*|$. Although this is theoretically slower (by a poly-logarithmic factor) than the algorithms in [8,16], it is designed specifically for STSP support graphs and runs very quickly in practice.

Once the cactus of G^* has been computed, we use it to construct a family of 'useful' tight sets. The following proposition provides the basis for our choice of which tight sets to include in our family:

Proposition 2 *A tour $\bar{x}$ is compatible with x^* if and only if the following two conditions hold:*

1. $\bar{x}(\delta(S)) = 2$ *whenever S is the bead of some circular partition;*
2. $\bar{x}_{uv} = 0$ *for each edge $\{u, v\}$ such that u and v appear in non-consecutive beads of a non-degenerate circular partition.*

Proof Let $\bar{x}$ be a tour vector that satisfies the second condition, and let $S_1, \ldots, S_q$ be the beads of a non-degenerate circular partition. The simultaneous equations

$$\bar{x}(\delta(S_k)) = 2 \qquad (k = 1, \ldots, q)$$

and

$$\sum_{i \in S_k, j \in S_{k+1}} \bar{x}_{ij} = 0 \qquad (k = 1, \ldots, q)$$

imply that

$$\sum_{i \in S_k, j \in S_{k+1}} \bar{x}_{ij} + \sum_{i \in S_k, j \in S_{k-1}} \bar{x}_{ij} = 2$$

for $k = 1, \ldots, q$. Together with the SECs, this implies the union of any set of consecutive beads is tight. Thus, $\bar{x}$ is compatible with x^*. The converse implication is similar. $\square$

This means that we only need to generate $\mathcal{O}(n)$ tight sets, one for each bead. These sets are constructed using a simple depth-first traversal of the cactus. During the same traversal, we compute the set of edges that can be fixed to zero via Proposition 2.

4.3 Phase 3: branch-and-cut

The final phase is to perform branch-and-cut in order to find a best compatible tour. Proposition 2 in the previous subsection suggests that one could simply take an existing branch-and-cut algorithm for the STSP, and modify it in two ways: delete every variable that corresponds to an edge whose end-vertices appear in non-consecutive beads of a non-degenerate circular partition, and insert a set of equations, to force $\mathcal{O}(n)$ sets to be tight.

Originally, we hoped to modify the branch-and-cut algorithm of `Concorde` in this way. Unfortunately, it does not appear to be possible in `Concorde` to insert additional equations into the formulation. We tried enforcing the equations via Lagrangian relaxation instead—by adding big penalties to the left-hand sides of the corresponding SECs—but this caused numerical instabilities. In the end, we decided to code our own branch-and-cut algorithm, drawing on `CPLEX` and `Concorde` routines when it is convenient to do so.

Since the branch-and-cut approach to the STSP is now well-established [1,17], we only explain the components of our algorithm that differ from the standard approach. These are as follows:

- The initial LP relaxation includes not only the degree equations and the trivial lower and upper bounds, but also one equation for each of the tight sets in our family.
- We do not perform variable pricing. Indeed, we observed that the number of variables that remains after deleting the variables mentioned above is usually quite small, making pricing unnecessary. More precisely, our edge-elimination rule typically enables us to eliminate around 95% of the edges from consideration. In Table 1 we show such a reduction for a subset of medium-size instances, namely between 500 and 1,000 vertices. The set is composed of 12 TSPLIB instances

Table 1 Effects of the edge-elimination rule on instances with $n \in [500, 1{,}000]$.

Name	$\lvert E \rvert$	$\lvert E' \rvert$	%ratio
att532	141,246	12,818	9.07
ali535	142,845	2,768	1.94
si535	142,845	4,713	3.30
pa561	157,080	1,746	1.11
u574	164,451	2,247	1.37
rat575	165,025	3,136	1.90
p654	213,531	962	0.45
d657	215,496	5,449	2.53
gr666	221,445	13,854	6.26
u724	261,726	2,525	0.96
rat783	306,153	2,115	0.69
dsj1000	499,500	4,253	0.85

(see Sect. 5) and Table 1 reports, for each instance, its name (name), number of initial edges ($|E|$), number of reduced edges ($|E'|$) and percentage ratio (%ratio) computed as $\frac{|E|}{|E'|} \cdot 100$.

- Before running branch-and-cut, we construct a feasible x^*-compatible tour, in order to have an initial upper bound. Note that we cannot run a standard STSP heuristic for this purpose, since it might yield a tour that is not x^*-compatible. Fortunately, an x^*-compatible tour can be easily generated, by taking an arbitrary Eulerian tour of the cactus.
- We decided to keep the cutting-plane component of our algorithm relatively simple. For this reason, we decided to separate only SECs, blossom inequalities and comb inequalities, in that order. We used the separation routines in `Concorde`.
- We decided to branch whenever the addition of cutting planes is not able to improve the bound by more than 10^{-3}. To branch, we simply choose the variable closest to 0.5. We explore always the subproblem with the smallest lower bound, i.e., we do pure best-bound first search. More precisely, at each branching decision, two nodes are created, the corresponding LPs are solved, and the nodes are stored by keeping the associated stack ordered by nonincreasing value of the lower bound.
- Every 10 cutting-plane iterations, we invoke reduced-cost fixing in an attempt to reduce the size of the LP.

5 Computational results

Our branch-and-cut algorithm was coded in C and run on a 2.40 GHz Intel Core2 PC, under a Linux operating system in x86_64 mode. As mentioned in the previous section, the code calls on functions from Wenger's cactus code, `Concorde` and `CPLEX`, and run using one core only.

We tested the algorithm on 103 STSP instances from the TSPLIB, the classical library of TSP instances described in Reinelt [18]. Our branch-and-cut algorithm is

Table 2 Finding a best compatible tour by branch-and-cut (part I)

Name	Best value	CPU time	Root %gap	Cactus time	B&C nodes	# cuts	SEP. time	TSP value	TSP %gap
burma14	3,323	0.00	0.00	0.00	0	0	0.00	3,323	0.00
att15	4,828	0.00	0.00	0.00	0	16	0.00	4,828	0.00
ulysses16	6,859	0.00	0.00	0.00	0	0	0.00	6,859	0.00
gr17	2,085	0.00	0.00	0.00	0	0	0.00	2,085	0.00
gr21	2,707	0.00	0.00	0.00	0	0	0.00	2,707	0.00
ulysses22	7,013	0.00	0.00	0.00	0	0	0.00	7,013	0.00
gr24	1,328	0.01	0.00	0.00	0	25	0.00	1,272	4.40
fri26	937	0.00	0.00	0.00	0	0	0.00	937	0.00
bayg29	1,610	0.00	0.00	0.00	0	15	0.00	1,610	0.00
bays29	2,039	0.00	0.00	0.00	0	19	0.00	2,020	0.94
dantzig42	699	0.02	0.00	0.00	0	23	0.01	699	0.00
swiss42	1,273	0.01	0.00	0.00	0	13	0.00	1,273	0.00
att48	10,653	0.02	0.28	0.00	2	18	0.01	10,628	0.24
gr48	5,226	0.02	0.00	0.00	0	60	0.01	5,046	3.57
hk48	11,525	0.01	0.00	0.00	0	47	0.01	11,461	0.56
eil51	448	0.01	0.00	0.00	0	37	0.00	426	5.16
berlin52	7,542	0.00	0.00	0.00	0	0	0.00	7,542	0.00
brazil58	25,395	0.01	0.00	0.00	0	11	0.00	25,395	0.00
st70	731	0.01	0.00	0.00	0	41	0.00	675	8.30
eil76	543	0.00	0.00	0.00	0	21	0.00	538	0.93
pr76	111,164	0.02	0.10	0.00	4	62	0.01	108,159	2.78
gr96	57,929	0.01	0.00	0.00	0	27	0.00	55,209	4.93
rat99	1,256	0.02	0.00	0.00	0	106	0.02	1,211	3.72
kroA100	21,767	0.02	0.00	0.00	0	80	0.01	21,282	2.28
kroB100	23,106	0.02	0.00	0.00	0	68	0.01	22,141	4.36

executed with a time limit of 3,600 CPU seconds. Computing times do not include the solution of the subtour relaxation (the SEP solution is an input for our procedure), i.e., Phase 1 described in the previous section.

The results are shown in Tables 2, 3, 4, 5. These tables report the following entries for each instance: the cost of the best compatible tour or the best one found within the time limit (best value, a "+" sign indicates if the time limit has been reached without an optimality proof), the CPU time (CPU time), the percentage gap at the root node with respect to the best compatible tour (root % gap), the computing time required to compute the cactus representation (cactus time), the number of branch-and-cut nodes excluding the root (B & C nodes), the overall number of cutting planes generated (# cuts) not including those generated to optimise over SEP, the separation time (sep. time), the optimal TSP tour value (TSP value) and the percentage gap between the cost of the best compatible tour and the cost of the optimal tour (%gap TSP, a "$\leq$" sign indicates that the gap is not greater than the indicated one). All computing times

Table 3 Finding a best compatible tour by branch-and-cut (part II)

Name	Best value	CPU time	Root %gap	Cactus time	B&C nodes	# cuts	SEP. time	TSP value	TSP %gap
kroC100	22,011	0.02	0.00	0.00	0	70	0.01	20,749	6.08
kroD100	21,792	0.02	0.00	0.00	0	78	0.01	21,294	2.34
kroE100	24,703	0.02	0.00	0.00	0	20	0.00	22,068	11.94
rd100	8,134	0.04	0.00	0.00	0	74	0.02	7,910	2.83
eil101	629	0.01	0.00	0.00	0	35	0.00	629	0.00
lin105	14,504	0.01	0.00	0.00	0	22	0.00	14,379	0.87
pr107	44,303	0.01	0.00	0.00	0	0	0.00	44,303	0.00
gr120	7,145	0.08	0.00	0.00	0	95	0.05	6,942	2.92
pr124	59,824	0.02	0.00	0.00	0	46	0.01	59,030	1.35
bier127	126,457	0.03	0.00	0.00	0	61	0.01	118,282	6.91
ch130	6,281	0.04	0.00	0.00	0	93	0.02	6,110	2.80
pr136	98,464	0.26	0.15	0.00	6	209	0.18	96,772	1.75
gr137	71,776	0.05	0.19	0.00	4	106	0.02	69,853	2.75
pr144	59,282	0.03	0.00	0.00	0	105	0.02	58,537	1.27
ch150	6,613	0.05	0.00	0.00	0	99	0.02	6,528	1.30
kroA150	28,090	0.04	0.00	0.00	0	75	0.01	26,524	5.90
kroB150	27,595	0.05	0.00	0.00	0	98	0.02	26,130	5.61
pr152	75,799	0.04	0.00	0.00	0	100	0.02	73,682	2.87
u159	43,152	0.04	0.00	0.00	0	55	0.01	42,080	2.55
si175	21,568	0.12	0.00	0.00	0	118	0.04	21,407	0.75
brg180	1,950	0.96	1.03	0.00	24	621	0.58	1,950	0.00
rat195	2,409	0.50	1.99	0.00	44	232	0.28	2,323	3.70
d198	15,966	0.08	0.08	0.00	2	129	0.03	15,780	1.18
kroA200	30,516	0.11	0.00	0.00	0	150	0.07	29,368	3.91
kroB200	32,165	0.06	0.00	0.00	0	109	0.02	29,437	9.27
gr202	40,326	0.05	0.00	0.00	0	54	0.01	40,160	0.41

are expressed in seconds. Note that the name of the instances immediately recalls the number of cities n of the problem. On two of the instances in Table 5, namely `u2152` and `u2319`, we are unable within the time limit to solve the problem to optimality. The table shows for those instances pretty large percentage gaps both at the root node and with respect to the optimal STSP tour. This is mainly due to the lack of a sophisticated primal heuristic. Indeed, running the code for a sufficiently long computing time we could prove that best compatible tours for `u2152` and `u2319` have value 66,317 and 239,737, respectively. Thus, the real percentage gaps are 0.89 and 1.43, respectively, (instead of 17.08 and 15.23 as in Table 5) at the root and 3.21 and 2.34, respectively, (instead of 23.36 and 19.00) with respect to the STSP optimal value. In other words, also for these instances, a best compatible tour is a 'good' approximation of the optimal STSP tour and the root node relaxation is rather tight.

Table 4 Finding a best compatible tour by branch-and-cut (part III)

Name	Best value	CPU time	Root %gap	Cactus time	B&C nodes	# cuts	SEP. time	TSP value	TSP %gap
ts225	146,110	0.03	0.00	0.00	0	104	0.01	126,643	15.37
tsp225	4,301	0.08	0.07	0.00	2	123	0.02	3,916	9.83
pr226	81,927	0.05	0.00	0.00	0	105	0.02	80,369	1.94
gr229	140,484	0.47	0.08	0.00	4	280	0.33	134,602	4.37
gil262	2,460	0.05	0.00	0.00	0	89	0.02	2,378	3.45
pr264	49,670	0.04	0.00	0.00	0	73	0.01	49,135	1.09
a280	2,784	0.07	0.14	0.00	12	110	0.04	2,579	7.95
pr299	50,871	0.05	0.00	0.00	0	82	0.02	48,191	5.56
lin318	42,736	0.09	0.00	0.00	0	115	0.03	42,029	1.68
rd400	15,704	0.24	0.00	0.00	0	226	0.16	15,281	2.77
fl417	12,122	0.12	0.04	0.00	4	112	0.04	11,861	2.20
gr431	178,367	0.62	0.00	0.00	0	318	0.33	171,414	4.06
pr439	109,320	0.48	0.00	0.01	0	278	0.27	107,217	1.96
pcb442	51,916	0.17	0.00	0.00	0	184	0.07	50,778	2.24
d493	36,828	0.26	0.00	0.00	0	220	0.09	35,002	5.22
att532	28,639	2.13	0.00	0.01	0	403	1.57	27,686	3.44
ali535	208,312	0.48	0.00	0.00	0	209	0.11	202,339	2.95
si535	48,572	16.02	0.10	0.01	414	565	5.68	48,450	0.25
pa561	2,950	0.37	0.00	0.01	0	300	0.16	2,763	6.77
u574	39,106	0.28	0.00	0.01	0	201	0.08	36,905	5.96
rat575	7,147	0.48	0.00	0.00	0	301	0.29	6,773	5.52
p654	34,833	0.21	0.05	0.00	2	150	0.05	34,643	0.55
d657	50,609	0.62	0.00	0.01	0	275	0.26	48,913	3.47
gr666	302,385	3.75	0.03	0.01	2	553	2.57	294,358	2.73
u724	43,944	0.51	0.01	0.01	6	291	0.21	41,910	4.85
rat783	9,082	0.42	0.00	0.01	0	232	0.12	8,806	3.13

Overall, the results reveal that the branch-and-cut algorithm performs rather well. Indeed, the computing times are usually small, exceeding 10 CPU seconds in only 11 cases. In addition, the number of branch-and-cut nodes is usually very small, exceeding 100 only in 5 cases. The computing time for constructing the cactus is always negligible, whereas separation consumes a substantial portion of the overall CPU time. Finally, the best compatible tour generally provides a good approximation of the optimal tour. On the one side, a compatible tour exploits structural properties of the problem (namely, an optimal SEP solution) by constructing one tour, thus – if used as a heuristic – it could be assimilated to the so-called “tour-construction” heuristics. If compared with 38 of those algorithms (results taken from [12]) on 26 TSPLIB instances between 1,000 and 6,000 cities, the %gap of a best compatible tour is better than the one of the best construction heuristic in 21 cases. On the other hand, finding a best compatible tour is $\mathcal{NP}$-hard and the computing times are generally not compa-

Table 5 Finding a best compatible tour by branch-and-cut (part IV)

Name	Best value	CPU time	Root %gap	Cactus time	B&C nodes	# cuts	SEP. time	TSP value	TSP %gap
dsj1000	19,608,590	1.46	0.00	0.01	0	535	0.71	18,660,188	5.08
pr1002	264,028	27.45	0.00	0.02	4	1,033	21.38	259,045	1.92
si1032	92,694	0.24	0.00	0.01	0	41	0.02	92,650	0.05
u1060	234,229	7.70	0.00	0.02	0	793	6.22	224,094	4.52
vm1084	252,438	1.00	0.09	0.02	6	409	0.30	239,297	5.49
pcb1173	60,468	0.61	0.00	0.02	0	227	0.08	56,892	6.29
d1291	52,735	1,806.89	0.57	0.03	1,886	3,709	319.95	50,801	3.81
rl1304	267,722	1.50	0.00	0.03	0	350	0.40	252,948	5.84
rl1323	299,529	0.86	0.00	0.02	0	233	0.08	270,199	10.85
nrw1379	58,573	44.73	0.23	0.03	10	1,136	26.16	56,638	3.42
fl1400	20,943	8.28	0.05	0.04	18	863	4.52	20,127	4.05
u1432	160,435	3.42	0.13	0.02	10	822	2.06	152,970	4.88
fl1577	23,661	1.14	0.00	0.02	0	273	0.16	22,249	6.35
d1655	72,327	2.02	0.00	0.05	0	416	0.39	62,128	3.80
vm1748	352,874	8.81	0.00	0.04	0	949	6.13	336,556	4.85
u1817	64,491	3.05	0.00	0.02	0	605	1.06	57,201	4.98
rl1889	333,610	7.64	0.00	0.04	2	823	4.82	316,536	5.39
d2103	83,146	3.32	0.00	0.04	0	410	1.30	80,450	3.35
u2152	$79{,}260^{+}$	3,600.00	17.08	0.07	112	3,950	1,774.33	64,253	≤ 23.36
u2319	$278{,}765^{+}$	3,600.00	15.23	0.06	52	4,391	2,325.28	234,256	≤ 19.00
pr2392	409,889	3.21	0.00	0.04	0	529	0.42	378,032	8.43
pcb3038	143,896	1,128.47	0.08	0.10	294	2,757	289.27	137,694	4.50
fl3795	30,656	29.12	0.07	0.13	134	1,148	14.94	28,772	6.55
fnl4461	1191,539	2,055.20	0.00	0.23	0	4,244	1,737.14	182,566	4.91
rl5915	595,401	301.82	0.07	0.43	56	2,394	142.35	565,530	5.28
rl5934	601,926	66.59	0.00	0.33	0	2,209	36.23	556,045	8.25

rable with those of the construction heuristics. It should be finally noticed that most of the "tour-improvement" heuristics, like for example Lin-Kernighan heuristic [13], clearly would do much better in the same amount of time.

At the end of the introduction, we mentioned the fact that the performance of the compatible tour heuristic could potentially be affected by the presence of alternative basic optimal solutions to the subtour relaxation. We did not find this to be a significant issue in our experiments, except in the case of the instance `ts225` (which was in fact designed to cause problems for LP-based techniques). To explore this issue, we experimented with the application of very small random perturbations to the objective function of `ts225`. In 1,000 random perturbations, we obtained several different 'best' compatible tours, with costs ranging from 128,775 to 146,110. These costs correspond to percentage gaps of between 1.68 and 15.37, with respect to the cost of the optimal tour.

Table 6 Comparing the effort for computing the compatible and optimal tours

Name	Compatible tour					Optimal STSP tour			
	Compatible value	Root %gap	B&C nodes	CPU time	SEP time	Optimal value	Root %gap	B&C nodes	CPU time
att532	28,639	0.00	0	2.13	0.27	27,686	0.25	67	94.06
ali535	208,312	0.00	0	0.48	0.56	202,339	0.12	9	13.25
si535	48,572	0.10	414	16.02	0.39	48,450	0.03	39	24.20
pa561	2,950	0.00	0	0.37	0.11	2,763	0.18	121	153.35
u574	39,106	0.00	0	0.28	0.17	36,905	0.06	11	26.99
rat575	7,147	0.00	0	0.48	0.08	6,773	0.13	103	119.63
p654	34,833	0.05	2	0.21	0.45	34,643	0.01	3	7.55
d657	50,609	0.00	0	0.62	0.20	48,913	0.25	239	406.59
gr666	302,385	0.03	2	3.75	0.58	294,358	0.13	27	42.13
u724	43,944	0.01	6	0.51	0.17	41,910	0.13	163	381.19
rat783	9,082	0.00	0	0.42	0.12	8,806	0.05	9	11.69
dsj1000	19,608,590	0.00	0	1.46	0.58	18,660,188	0.10	135	446.24

We conclude the computational section by giving evidence that finding a best compatible tour is much less expensive computationally than finding an optimal STSP tour. Specifically, we considered the 12 medium-size instances with $n \in [500, 1, 000]$ (already used in Sect. 4.3, Table 1) to compare the computing time and the branch-and-bound nodes needed by our code for the compatible tour and by `Concorde` to solve the STSP, respectively. To make the comparison fair, we limited the arsenal of the latter to the use of the same type of cuts we use for the computation of the former. (Recall that we use `Concorde` cuts only.) The results are given in Table 6 where for both the compatible tour and the STSP tour computations we report the optimal values, the root percentage gap, the computing time (in CPU seconds) and the number of branch-and-cut nodes. To allow a fully fair comparison, we also report the computing time to optimise over the SEP (SEP time) which should be added to the CPU time in the compatible tour computation. The results in Table 6 show that computing a best compatible tour is at least one order of magnitude faster than finding an optimal tour but for instance `si535` for which it is comparable. Such a difference can grow up to three orders of magnitude.

6 Concluding remarks

In this paper, we have defined compatible tours for the STSP, derived several theoretical results concerning them, and presented extensive computational results. Although the theoretical results are of a negative nature, the computational results are very encouraging. More precisely, the time taken to compute a best compatible tour is reasonable in practice, and the tour found is frequently of extremely good quality.

In particular, the fact that finding a best compatible tour is so much faster than finding a best tour is interesting and somewhat surprising. Thus, one can foresee the idea of using the computation of a compatible tour in different contexts with respect to the heuristic one. Indeed, an interesting topic for future research would be the exploitation of the concept of compatible tours either for generating cutting planes or for devising new branching rules. For example, one could seek cutting planes that are not only valid, but also satisfied at equality by at least one compatible tour, or by all compatible tours, in a sort of *primal* fashion. Or one could branch by using disjunctions of the form $(x(\delta(S)) = 2) \vee (x(\delta(S)) \geq 2)$, where S is a bead in some circular partition, or more complex disjunctions based on the circular partitions themselves.

Finally, an important open question is to determine the worst-case ratio between the costs of the best compatible tour and the optimal tour, when the edge weights satisfy the triangle inequality. We have shown that this ratio can approach 5/3, but we do not know if this bound is tight. Another important open question is, of course, whether the integrality gap of the subtour relaxation is 4/3 under the same condition [10].

Acknowledgments The authors are grateful to Sylvia Boyd and Martin Savelsbergh for useful discussions. We warmly thank two anonymous referees and the editors for very useful comments which helped to clarify and improve the paper.

References

1. Applegate, D., Bixby, R.E., Chvátal, V., Cook, W.: The Traveling Salesman Problem: a Computational Study. Princeton University Press, Princeton (2007)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comp. Sys. Sci. **12**, 335–379 (1976)
3. Boyd, S., Pulleyblank, W.R.: Optimizing over the subtour polytope of the traveling salesman problem. Math. Program **49**, 163–187 (1991)
4. Burkhard, R.E., Deineko, V.G., Woeginger, G.J.: The travelling salesman and the PQ-tree. Math. Oper. Res. **23**, 613–623 (1998)
5. Concorde TSP Solver. http://www.tsp.gatech.edu/concorde.html
6. Deineko, V., Tiskin, A.: Fast minimum-weight double-tree shortcutting for metric TSP. In: Demetrescu, C. (ed.) Experimental Algorithms. Lecture Notes in Computer Science, Springer, Berlin (2007)
7. Dinitz, E.A., Karzanov, A.V., Lomonosov, M.V.: A structure for the system of all minimum cuts of a graph. In: Fridman, A.A. (ed.) Studies in Discrete Optimization, Nauka, Moscow (In Russian) (1976)
8. Fleischer, L.: Building chain and cactus representations of all minimum cuts from Hao-Orlin in the same asymptotic run time. J. Algorithms **33**, 51–72 (1999)
9. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified $\mathcal{NP}$-complete problems. In Proceedings of the 6th Annual ACM Symposium on Theory of Computing. ACM Press, New York (1974)
10. Goemans, M.X.: Worst-case comparison of valid inequalities for the TSP. Math. Program. **69**, 335–349 (1995)
11. Gutin, G., Punnen, A.P. (eds.): The Traveling Salesman Problem and its Variations. Kluwer, Dortrecht (2002)
12. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, P. (eds.) op. cit. http://www.research.att.com/~dsj/chtsp/ (2002)
13. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math. Program. Computation **1**, 119–163 (2009)
14. Lawler, E., Lenstra, J., Rinnooy Kan, A., Shmoys, D. (eds.): The Traveling Salesman Problem. Wiley, Chichester
15. Letchford, A.N., Pearson, N.A.: Good triangulations yield good tours. Comp. Oper. Res. **35**, 638–647 (2008)
16. Nagamochi, H., Nakamura, S., Ishii, T.: Constructing a cactus for minimum cuts of a graph in $O(mn+n^2 \log n)$ and $O(m)$ space. IEICE Trans. Inf. Sys. **E86-D**, 179–185 (2003)

17. Padberg, M.W., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems. SIAM Rev. **33**, 60–100 (1991)
18. Reinelt, G.: TSPLIB — a traveling salesman problem library. ORSA J. on Computing **3**, 376–384 (1991)
19. De Vitis, A.: The cactus representation of all minimum cuts in a weighted graph. Technical Report 454, IASI-CNR, Rome (1997)
20. Wenger, K.M.: A new approach to cactus construction applied to TSP support graphs. In: Cook, W., Schulz, A.S. (eds.) Integer Programming and Combinatorial Optimization IX. Lecture Notes in Computer Science, vol. 2337, Springer, Berlin (2002)
21. Wenger, K.M.: Generic Cut Generation Methods for Routing Problems. Shaker Verlag, Aachen (2004)