Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

EXACT METHODS FOR THE ASYMMETRIC TRAVELING SALESMAN PROBLEM

Matteo Fischetti

D.E.I, University of Padova

Via Gradenigo 6/A, 35100 Padova, Italy

fisch@dei.unipd.it

(2013)

Abstract

In the present chapter we concentrate on the exact solution methods for the Asymmetric *TSP* proposed in the literature after the writing of the survey of Balas and Toth [81]. In Section 2 two specific branch-and-bound methods, based on the solution of the assignment problem as a relaxation, are presented and compared. In Section 3 a branch-and-bound method based on the computation of an additive bound is described, while in Section 4 a branch-and-cut approach is discussed. Finally, in Section 5 all these methods are computationally tested on a large set of instances, and compared with an effective branch-and-cut code for the symmetric *TSP*.

A formal definition of the problem is as follows. Let $G = (V, A)$ be a given complete digraph, where $V = \{1, \ldots, n\}$ is the vertex set and

$A = \{(i,j) : i,j \in V\}$ the arc set, and let c_{ij} be the cost associated with arc $(i,j) \in A$ (with $c_{ii} = +\infty$ for each $i \in V$). A *Hamiltonian directed cycle* (*tour*) of G is a directed cyclevisiting each vertex of V exactly once, i.e., a spanning subdigraph $\tilde{G} = (V, \tilde{A})$ of G such that $|\tilde{A}| = n$, and $\tilde{G}$ is strongly connected, i.e., for each pair of distinct vertices $i,j \in V$, $i < j$, both paths from i to j and from j to i exist in $\tilde{G}$.

The *Asymmetric Traveling Salesman Problem (ATSP)* is to find a Hamiltonian directed cycle $G^* = (V, A^*)$ of G whose *cost* $\sum_{(i,j)\in A^*} c_{ij}$ is a minimum. Without loss of generality, we assume $c_{ij} \geq 0$ for any arc $(i,j) \in A$.

The following Integer Linear Programming formulation of *ATSP* is well-known:

$$v(ATSP) = \min \sum_{(i,j)\in A} c_{ij}\, x_{ij} \tag{1}$$

subject to

$$\sum_{i\in V} x_{ij} = 1 \qquad j \in V \tag{2}$$

$$\sum_{j\in V} x_{ij} = 1 \qquad i \in V \tag{3}$$

$$\sum_{i\in S} \sum_{j\in V\setminus S} x_{ij} \geq 1 \qquad S \subset V : S \neq \emptyset \tag{4}$$

$$x_{ij} \geq 0 \qquad i,j \in V \tag{5}$$

$$x_{ij} \text{ integer} \qquad i,j \in V \tag{6}$$

where $x_{ij} = 1$ if and only if arc (i,j) is in the optimal tour. Constraints (2) and (3) impose the in-degree and out-degree of each vertex be equal to one, respectively, while constraints (4) impose strong connectivity. Because of (2) and (3), conditions (4) can be equivalently re-written as the *Subtour Elimination Constraints* (SECs):

$$\sum_{i\in S} \sum_{j\in S} x_{ij} \leq |S| - 1 \quad S \subset V : S \neq \emptyset \tag{7}$$

Moreover, it is well known that one can halve the number of constraints (4) by replacing them with

$$\sum_{i\in S} \sum_{j\in V\setminus S} x_{ij} \geq 1 \quad S \subset V : r \in S \tag{8}$$

or with

$$\sum_{i\in S} \sum_{j\in V\setminus S} x_{ij} \geq 1 \quad S \subset V : S \neq \emptyset, r \notin S \tag{9}$$

where r is any fixed vertex.

Several substructures of *ATSP* can be pointed out, each associated with a subset of constraints defining a well-structured relaxation whose solution value gives a valid lower bound for *ATSP*.

Constraints (2), (3) and (5), with objective function (1), define the well-known *min-sum Assignment Problem* (*AP*). Such a problem always has an integer optimal solution, and requires finding a minimum-cost collection of vertex-disjoint *subtours* visiting all the vertices of *G*. If an optimal solution of *AP* determines only one directed cycle, then it satisfies all constraints (4) and hence is optimal for *ATSP* as well. Otherwise, each vertex subset *S* whose vertices are visited by the same subtour, determines a violated constraint (4). Relaxation *AP* can be solved in $O(n^3)$ time (see, e.g., Lawler [547]).

Constraints (2), (8) and (5), with objective function (1), define the well-known shortest *Spanning r-Arborescence Problem* (r-*SAP*). Such a problem always has an integer optimal solution, and corresponds to finding a minimum-cost spanning subdigraph $\overline{G} = (V, \overline{A})$ of G such that (i) the in-degree of each vertex is exactly one, and (ii) each vertex can be reached from the *root* vertex r. If an optimal solution of r-*SAP* leaves each vertex with out-degree equal to one, then it satisfies all constraints (3) and hence is optimal for *ATSP* as well. Otherwise, each vertex having out-degree different from one, determines a violated constraint (3). Relaxation r-*SAP* can be solved in $O(n^2)$ time by finding the shortest spanning arborescence rooted at vertex r, and by adding to it a minimum-cost arc entering vertex r. Efficient algorithms for the shortest arborescence problem have been proposed by Edmonds [267], Fulkerson [338], Tarjan [788], and Camerini, Fratta and Maffioli [154, 155]; an efficient implementation of Tarjan's algorithm can be found in Fischetti and Toth [305]. Fischetti [294] described a modified $O(n^2)$-time method to compute an improved lower bound not depending on the root vertex r.

A third substructure, corresponding to constraints (3), (9) and (5), with objective function (1), defines the shortest *Spanning r-Antiarborescence Problem* (r-*SAAP*). Such a problem can easily be transformed into r-*SAP* by simply transposing the input cost matrix, hence it can be solved in $O(n^2)$ time.

In order to obtain tighter lower bounds, two enhanced relaxations, r-*SADP* and r-*SAADP*, can be introduced.

Relaxation r-*SADP* is obtained from r-*SAP* by adding constraint (3) for $i = r$, i.e., $\sum_{j \in V} x_{rj} = 1$, which imposes out-degree equal to one for the root vertex r. Such a problem can be transformed into r-*SAP* (and hence solved in $O(n^2)$ time) by considering a modified cost matrix obtained by adding a large positive value M to costs c_{rj} for all $j \in V$, the optimal value of r-$SADP$ being $v(r\text{-}SAP) - M$.

Relaxation r-$SAADP$ is obtained in a similar way from r-$SAAP$, by adding constraint (2) for $j = r$, i.e., $\sum_{i \in V} x_{ir} = 1$, which imposes in-degree equal to one for the root vertex r. Such a problem can be solved in $O(n^2)$ time by transforming it into r-$SADP$ through transposition of the input cost matrix.

2. *AP*-Based Branch-and-Bound Methods

In this section we review the *AP*-based branch-and-bound algorithms that have been proposed since the writing of the Balas and Toth [81] survey. All these algorithms are derived from the lowest-first branch and bound procedure TSP1 presented in Carpaneto and Toth [166] which is outlined below.

At each node h of the decision tree, procedure TSP1 solves a *Modified Assignment Problem* (MAP_h) defined by (1), (2), (3), (5) and the additional variable-fixing constraints associated with the following arc subsets:

$$
\begin{aligned}
E_h &= \{(i,j) \in A : x_{ij} \text{ is fixed to } 0\} \quad \text{(excluded arcs)} \\
I_h &= \{(i,j) \in A : x_{ij} \text{ is fixed to } 1\} \quad \text{(included arcs)}
\end{aligned}
$$

MAP_h can easily be transformed into a standard *AP* by properly modifying the cost matrix so as to take care of the additional constraints.

If the optimal solution to MAP_h does not define a Hamiltonian directed cycle and its value LB_h (yielding the lower bound associated with node h) is smaller than the current optimal solution value, say *UB,* then m descending nodes are generated from node h according to the following branching scheme (which is a modification of the subtour elimination rule proposed by Bellmore and Malone [97]).

Let $G_p = (V_p, A_p)$ be a subtour in the optimal MAP_h solution having the minimum number of not included arcs, i.e., such that $m := |A_p \backslash I_h|$ is a minimum, and let $(s_1, t_1), \ldots, (s_m, t_m)$ be the non-included arcs of A_p taken in the same order as they appear along the subtour. The subsets of the excluded/included arcs associated with the j-th descending node of the current branching node h, say $g(j)$, are defined as follows ($j = 1, \ldots, m$) (see also Figure 4.1 for an illustration):

$$
\begin{aligned}
E_{g(j)} &= E_h \cup \{(s_j, t_j)\} \\
I_{g(j)} &= I_h \cup \{(s_i, t_i) : i = 1, \ldots, j-1\}
\end{aligned}
$$

Moreover, each subset $E_{g(j)}$, $j > 1$, is enlarged by means of the arc (t_{j-1}, s_1), so as to avoid subtours with just one non-included arc.

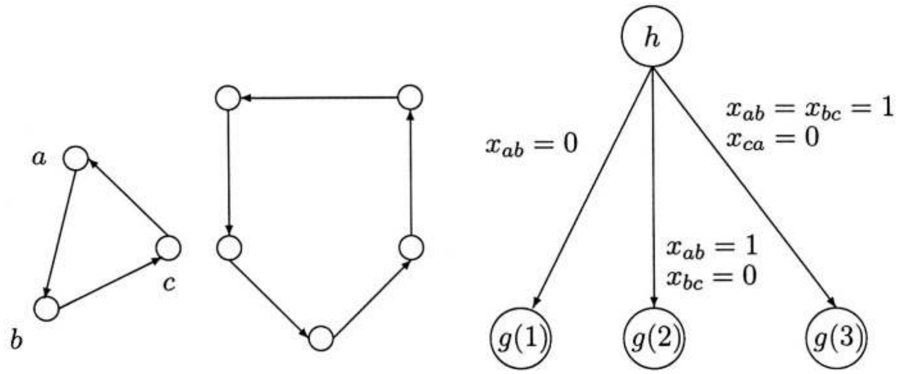

Figure 4.1. Branching on subtour $G_p = (V_p, A_p)$, where $V_p = \{a, b, c\}$ and $A_p = \{(a, b), (b, c), (c, a)\}$.

2.1. The Algorithm of Carpaneto, Dell'Amico and Toth

The approach of Carpaneto, Dell'Amico and Toth [164] differs from that presented in [166] in the following main respects:

a) at the root node of the branch-decision tree, application of a *reduction procedure* to remove from G some arcs that cannot belong to an optimal tour; in this way the original digraph G can be transformed into a sparse one, say $\widetilde{G} = (V, \widetilde{A})$, allowing the use of procedures specialized for sparse graphs;

b) use of an efficient *parametric technique* for the solution of the *MAP's,* allowing each MAP_h to be solved in $O(|\widetilde{A}| \log n)$ time;

c) application, at each branching node h, of a *subtour merging* procedure to decrease the number of subtours defined by the optimal MAP_h solution.

2.1.1 Reduction Procedure. At the root node of the branch-decision tree, the *AP* corresponding to the original complete cost matrix, c, is solved through the $O(n^3)$ primal-dual procedure CTCS presented in Carpaneto and Toth [168]. Let (u_i) and (v_j) be an optimal solution of the dual problem associated with *AP,* and let LB_0 be the corresponding solution value. It is well known that, for each arc $(i, j) \in A$, the *reduced cost* $\bar{c}_{ij} = c_{ij} - u_i - v_j \geq 0$ represents a lower bound on the increase of the optimal *AP* solution value corresponding to the inclusion of arc (i, j). If an *ATSP* feasible solution of value *UB* is known, then each arc $(i, j) \in A$ such that $\bar{c}_{ij} \geq UB - LB_0$ can be removed from *A,* since its inclusion in any *ATSP* solution cannot lead to a solution value smaller than *UB*. The original complete digraph *G* can thus be reduced into a sparse one, $\widetilde{G} = (V, \widetilde{A})$, where $\widetilde{A} = \{(i,j) \in A : \bar{c}_{ij} < UB - LB_0\}$.

Value *UB* can be obtained through any heuristic procedure for *ATSP*; in [164] the patching algorithm proposed by Karp [498] is used. An alternative way is to compute an "artificial" upper bound by simply setting $UB = \alpha\ LB_0$, where $\alpha > 1$ is a given parameter. However, if at the end of the branch and bound algorithm no feasible solution of value less than *UB* is found, then $\alpha\ LB_0$ is not a valid upper bound, so α must be increased and a new run needs to be performed.

2.1.2 Parametric *MAP* Solution. The effectiveness of the overall *ATSP* algorithm greatly depends on the efficiency of the *MAP* algorithm used. At each node h of the decision tree, instead of solving MAP_h from scratch, a parametric technique is used which finds only one shortest augmenting path. Indeed, when generating a descending node h from its father node k, only one arc, say (s,t), is excluded from the solution of MAP_k. So, to obtain the optimal solution of MAP_h from that of MAP_k it is only necessary to satisfy constraint (2) for $j = t$ and constraint (3) for $i = s$, i.e., one only needs to find a single shortest augmenting path from vertex s to vertex t in the bipartite graph corresponding to MAP_h with respect to the current reduced cost matrix $\bar{c}$. Note that the addition of the new included arcs contained in $I_h \backslash I_k$ does not affect the parametrization, as these arcs already belong to the optimal solution of MAP_k. As graph $\widetilde{G}$ is sparse, the shortest augmenting path is found through a procedure derived from the labelling algorithm proposed by Johnson [459] for the computation of shortest paths in sparse graphs, which uses a heap queue. Hence, the resulting time complexity for solving each MAP_h is $O(|\widetilde{A}|\ log\ n)$.

The computation of the shortest augmenting path at each node h is stopped as soon as its current reduced cost becomes greater or equal to the gap between the current upper bound value UB and the optimal value of MAP_k.

2.1.3 Subtour Merging. Consider a node h of the decision tree for which several optimal *MAP* solutions exist. Computational experience shows that the optimal solution which generally leads to the smallest number of nodes in the subtree descending from h is that having the minimum number of subtours. A heuristic procedure which tries to decrease the number of subtours is obtained by iteratively applying the following rule.

Given two subtours $G_a = (V_a, A_a)$ and $G_b = (V_b, A_b)$, if there exists an arc pair $(i_a, j_a) \in A_a$ and $(i_b, j_b) \in A_b$ such that arcs (i_a, j_b) and (i_b, j_a) have zero reduced costs (i.e., $\overline{c}_{i_a j_b} = \overline{c}_{i_b j_a} = 0$), then an equivalent optimal solution to MAP_h can be obtained by connecting subtours G_a and G_b to form a single subtour $G_a = (V_a \cup V_b, A_a \cup A_b \backslash \{(i_a, j_a), (i_b, j_b)\} \cup \{(i_a, j_b), (i_b, j_a)\})$. If, at the end of the procedure, a Hamiltonian directed cycle is found, then an optimal solution to the *ATSP* associated with node h has been found and no descending nodes need to be generated.

The above subtour-merging procedure is alwaysapplied at the root node of the decision tree. As to the other nodes, it is applied only if the total number of zero reduced cost arcs at the root node is greater than a given threshold β (e.g., $\beta = 2.5n$). Indeed, the procedure is typically effective only if the subdigraph corresponding to the zero reduced-cost arcs contains a sufficiently large number of arcs. (Computational experiments have shown that an adaptive strategy, which counts the number of zero reduced-cost arcs at each node and then decides on the opportunity to apply the procedure, often gives worseresults than the threshold method above.)

2.2. The Algorithms of Miller and Pekny

Effective procedures for the solution of the *ATSP* have been proposed by Miller and Pekny in the early nineties [596, 597, 665, 664]. These methods are also based on the general approach presented in Carpaneto and Toth [166], the main differences and similarities between them being discussed below.

In [596], Miller and Pekny presented a preliminary algorithm which is a parallelization of the approach of Carpaneto and Toth, improved with the application of the patching heuristic [498] at the root node.

The algorithm presented in [664] represents a substantial improvement of the original parallel procedure. The *MAP*'s at the nodes are solved through an $O(n^2)$ parametric procedure which computes a single augmenting path using a *d-heap*. Moreover, the patching algorithm is applied at the root node, and to the other nodes with decreasing frequency as search progresses. In addition, the branch-and-bound phase is preceded by a sparsification of the cost matrix obtained by removing all the entries with cost greater than a given threshold λ. Asufficient condition is given to check whether the optimal solution obtained with respect to the sparse matrix is optimal for the original matrix as well.

The algorithm presented in [665] is a modification of that presented in [664], obtained with the application, at each node, of an exact procedure to find a Hamiltonian directed cycle on the subdigraph defined by the arcs with zero reduced cost. The most sophisticated version of the Miller and Pekny codes appears to be that presented in [597], which includes all the improvements previously proposed by the authors.

The similarities among the approach of Carpaneto, Dell'Amico and Toth [164] and the algorithms of Miller and Pekny are the following: (a) the branching rule is that proposed in [166]; (b) the *MAP*'s at the various branching nodes are solved through an $O(n^2)$ procedure; and (c) the patching algorithm is applied at the root node. The two approaches differ in the following aspects: (a) for the sparsification phase, [164] proposes a criterion based on the comparison between the reduced costs given by the initial *AP* and the gap between lower and upper bound; (b) an efficient technique to store and retrieve the subproblems is proposed in [164] so that the exploration of the branch-decision tree is accelerated; and (c) a fast heuristic algorithm to find a Hamiltonian directed cycle on the subdigraph defined by the arcs with zero reduced cost is applied in [164].

Comparing the computational results obtained by Miller and Pekny with those presented in [164], it appears that the latter code is slower than the algorithm presented in [597] for small cost ranges (and random instances), but it seems to be faster for large cost ranges. On the whole, the two methods exhibit a comparable performance.

3. An Additive Branch-and-Bound Method

This section describes the solution approach proposed by Fischetti and Toth [304], who embedded a more sophisticated bounding procedure within the standard branch-and-bound method of Carpaneto and Toth [166]. Observe that *AP*, r-$SADP$ and r-$SAADP$ relaxations (as defined in Section 1) are complementary to each other. Indeed, *AP* imposes the degree constraints for all vertices, while connectivity constraints are completely neglected. Relaxation r-$SADP$, instead, imposes reachability from vertex r to all the other vertices, while out-degree constraints are neglected for all vertices different from r. Finally, r-$SAADP$ imposes reachability from all the vertices to vertex r, while in-degree constraints are neglected for all vertices different from r. A possible way of combining the three relaxations is to apply the so-called *additive approach* introduced by Fischetti and Toth [303].

3.1. An Additive Bounding Procedure

An additive bounding procedure for *ATSP* can be outlined as follows. Let $\mathcal{L}^{(1)}, \mathcal{L}^{(2)}, \ldots, \mathcal{L}^{(q)}$ be q bounding procedures available for *ATSP*. Suppose that, for $h = 1, 2, \ldots, q$ and for any cost matrix $\bar{c}$, procedure $\mathcal{L}^{(h)}(\bar{c})$, when applied to the *ATSP* instance having cost matrix $\bar{c}$, returns its lower bound $\delta^{(h)}$ as well as a so-called *residual cost* matrix $c^{(h)}$ such that:

i) $c_{ij}^{(h)} \geq 0$ for each $i, j \in V$;

ii) $\delta^{(h)} + \sum_{(i,j)\in A} c_{ij}^{(h)} x_{ij} \leq \sum_{(i,j)\in A} \bar{c}_{ij} x_{ij}$ for each feasible *ATSP* solution x.

The additive approach generates a sequence of *ATSP* instances, each obtained by considering the residual cost matrix corresponding to the previous instance and by applying a different bounding procedure. A Pascal-like outline of the approach follows.

ALGORITHM ADDITIVE:
1. **input**: cost matrix c;
2. **output**: lower bound δ and the residual-cost matrix $c^{(q)}$;

begin

3. initialize $c^{(0)} := c,\ \delta := 0$;
4. **for** $h := 1$ **to** q **do**

 begin

5. apply $\mathcal{L}^{(h)}(c^{(h-1)})$, thus obtaining value $\delta^{(h)}$ and the residual cost matrix $c^{(h)}$;
6. $\delta := \delta + \delta^{(h)}$

 end

end.

An inductive argument shows that the values δ computed at step 6 give a non decreasing sequence of valid lower bounds for *ATSP*. Moreover, the final residual-cost matrix $c^{(q)}$ can be used for reduction purposes.

Related approaches, using reduced costs for improving lower bounds for *ATSP*, are those of Christofides [188] and Balas and Christofides [70]. For a comparison of the additive approach with the restricted Lagrangian approach of Balas and Christofides, the reader is referred to [304].

Note that, because of condition ii) above, each bounding procedure $\mathcal{L}^{(h)}$ of the sequence introduces an *incremental gap*

$$\gamma^{(h)} = v^{(h-1)}(ATSP) - (v^{(h)}(ATSP) + \delta^{(h)}) \geq 0,$$

where $v^{(k)}(ATSP)$ denotes the optimal solution value of the *ATSP* instance associated with cost matrix $c^{(k)}$. It follows that the overall gap γ between $v(ATSP)$ and the final lower bound $\delta = \sum_{h=1}^{q} \delta^{(h)}$ cannot be less than $\sum_{h=1}^{q} \gamma^{(h)}$.

Procedures $\mathcal{L}^{(1)}, \mathcal{L}^{(2)}, \ldots, \mathcal{L}^{(q)}$ can clearly be applied in a different sequence, thus producing different lower bound values and residual costs. As a heuristic rule, it is worthwhile to apply procedures $\mathcal{L}^{(h)}$ according to increasing (estimated) percentage incremental gaps, so as to avoid the introduction of high percentage incremental gaps at the beginning of the sequence, when the current value $v^{(h)}(ATSP)$ is still large.

A key step of the above algorithm is the computation of the residual costs. Since all the bounding procedures considered in [304] are based on linear programming relaxations, valid residual cost matrices can be obtained by computing the *reduced cost* matrices associated with the corresponding *LP*-dual optimal solutions.

Reduced costs associated with the *AP* relaxation can easily be obtained without extra computational effort. As to the reduced costs for r-SAP, those associated with the arcs not entering the root vertex r are the reduced costs of the shortest spanning arborescence problem (which can be computed in $O(n^2)$ time through a procedure given in [305]), while those associated with the arcs entering r are obtained by subtracting their minimum from the input costs. Reduced costs for problems r-$SAAP$, r-$SADP$ and r-$SAADP$ can be obtained in a similar way.

Here is an overall additive bounding algorithm, subdivided into four stages.

```
ALGORITHM ADD-ATSP:
1. input:   cost matrix c;
2. output:  lower bound δ and the residual-cost matrix c̄;
   begin
Stage 1:
3.     solve problem AP on the original cost matrix c and
       let c̄ be the corresponding reduced cost matrix;
       δ := v(AP);
Stage 2:
4.     solve problem 1-SAP on cost matrix c̄ and update
       c̄ to become the corresponding reduced cost matrix;
       δ := δ + v(1-SAP);
5.     solve problem 1-SAAP on cost matrix c̄ and update
       c̄ to become the corresponding reduced cost matrix;
       δ := δ + v(1-SAAP);
```

```
Stage 3:
6.    for r := 1 to n do
        begin
7.        solve problem r-SADP on cost matrix c̄ and
          update c̄ to become the corresponding reduced
          cost matrix;
          δ := δ + v(r-SADP)
        end;
Stage 4:
8.    for r := 1 to n do
        begin
9.        solve problem r-SAADP on cost matrix c̄ and
          update c̄ to become the corresponding reduced
          cost matrix;
          δ := δ + v(r-SAADP)
        end
  end.
```

Let $G_0 = (V, A_0)$ denote the spanning subdigraph of G defined by the arcs whose current reduced cost is zero.

At Stage 1, the bounding procedure based on the *AP* relaxation is applied. After this stage, each vertex in G_0 has at least one entering and leaving arc; however, G_0 is not guaranteed to be strongly connected.

At Stage 2, one forces the strong connectivity of G_0 by applying the bounding procedures based on 1-*SAP* and 1-*SAAP*. Indeed, after step 4 each vertex can be reached from vertex 1 in G_0, whereas after step 5 each vertex can reach vertex 1.

The current spanning subdigraph G_0 may at this point contain a tour, in which case lower bound δ cannot be further increased through an additive approach. If such a tour has been detected, it corresponds to a heuristic solution to *ATSP,* whose optimality can be checked by comparing its original cost with lower bound δ. More often, however, spanning subdigraph G_0 is non-Hamiltonian.

Let the forward and backward star of a node h in a given digraph $G' = (V', A')$ be defined as $\{j \in V' : (h, j) \in A'\}$ and $\{i \in V' : (i, h) \in A'\}$, respectively. We say that a vertex $r \in V$ is a *forward articulation point* of G_0 if none of the vertices of its forward star can reach all other vertices in $V \setminus \{r\}$ without passing through vertex r. Analogously, a vertex $r \in V$ is said to be a *backward articulation point* of G_0 if none of the vertices of its backward star can be reached from all the other vertices in $V \setminus \{r\}$ without passing through vertex r. Clearly, the existence of a forward or backward articulation point is a sufficient condition for G_0 to be non-Hamiltonian.

A related concept is that of (undirected) articulation point: a vertex r is an articulation point of G_0 if the underlying undirected subdigraph of G_0 induced by vertex subset $V \setminus \{r\}$ has more than one connected component. Notice that concept of forward (or backward) articulation is stronger than that of (undirected) articulation. Indeed, if vertex r is an (undirected) articulation point of G_0, then it is also a forward and a backward articulation point of G_0, while the opposite does not always hold.

The existence of a forward (resp. backward) articulation point r of G_0 can be exploited to increase the current lower bound δ by solving relaxation r-*SADP* (resp. r-*SAADP*). Indeed, in this case no zero cost r-arborescence (resp. r-antiarborescence) in which r has out-degree (resp. in-degree) equal to one, exists with respect to the current reduced costs $\bar{c}_{ij}$. Accordingly, for each vertex r one applies bounding procedures based on relaxations r-*SADP* (at Stage 3) and r-*SAADP* (at Stage 4), so as to increase the current lower bound δ in case the root vertex r is a forward or backward articulation point, respectively. After each execution of steps 7 and 9, the current vertex r is guaranteed not to be a forward or backward articulation point of the current graph G_0, respectively.

The overall time complexity of algorithm ADD-ATSP is $O(n^3)$, the most time-consuming steps being step 3 ($O(n^3)$), and steps 7 and 9 ($O(n^2)$), which are executed n times.

The tightness of the final lower bound δ greatly depends on the reduced costs obtained after each lower bound computation. In particular, consider the *LP*-dual of *AP*, defined by:

$$\begin{aligned} v(D\text{-}AP) &= \max \sum_{i \in V} (u_i + v_i) \\ &\text{subject to} \\ c_{ij} - u_i - v_j \geq 0 & \qquad i, j \in V \end{aligned}$$

and let $(\bar{u}, \bar{v})$ be the dual optimal solution found at step 3. For each vertex $h \in V$, let $\bar{L}_h$ be the cost of the shortest path from vertex 1 to vertex h, computed with respect to the current reduced costs $\bar{c}_{ij} = c_{ij} - \bar{u}_i - \bar{v}_j$. It is known (see, e.g., [505]) that an alternative dual optimal solution (u^*, v^*) is given by $u^*_i = \bar{u}_i - \bar{L}_i$ and $v^*_i = \bar{v}_i + \bar{L}_i$ for each $i \in V$. (Indeed, one has $\sum_{i \in V} (u^*_i + v^*_i) = \sum_{i \in V} (\bar{u}_i + \bar{v}_i)$ while, for each $j \in V$, $c_{ij} - u^*_i - v^*_j = \bar{c}_{ij} + \bar{L}_i - \bar{L}_j \geq 0$ follows from the definition of $\bar{L}_h$'s as costs of shortest paths.) Now, let $c^*_{ij} = c_{ij} - u^*_i - v^*_j$ be the reduced costs associated with this alternative dual optimal solution. One can easily verify that the cost P^*_{hk} of any simple path from vertex h to vertex k (computed with respect to c^*), is equal to $\overline{P}_{hk} + \overline{L}_h - \overline{L}_k$, where $\overline{P}_{hk}$ is the cost of the same path computed with respect to $\overline{c}$. Therefore, c^* can be viewed as a biased reduced-cost matrix obtained from $\overline{c}$ by reducing the cost of the paths emanating from vertex 1, while increasing the cost of the paths towards vertex 1.

A new additive bounding algorithm, `B-ADD-ATSP` (`B` for biased), can now be obtained from `ADD-ATSP` by adding the following step right after step 3:

3'. `compute (on the current reduced cost matrix` $\overline{c}$`) the cost` $\overline{L}_h$ `of the shortest path from vertex 1 to all vertices` $h \in V$`;`
for each $i, j \in V$ **do** $\overline{c}_{ij} := \overline{c}_{ij} + \overline{L}_i - \overline{L}_j$

Note that, after step 3', spanning subdigraph G_0 contains a 1-arborescence, hence step 4 can be omitted in `B-ADD-ATSP`.

At first glance, algorithm `B-ADD-ATSP` appears to be weaker than `ADD-ATSP`, since a step producing a possible increase on the current lower bound (step 4) has been replaced by a step which gives no improvement (step 3'). However, the cost biasing introduced at step 3' may allow the subsequent step 5 to increase its contribution to the current lower bound. Computational experience has shown that `B-ADD-ATSP` typically outperforms `ADD-ATSP`, hencealgorithm `B-ADD-ATSP` is chosen in [304].

As to the experimental computing time of algorithm `B-ADD-ATSP`, it can greatly be reduced by the implementation given in [304].

4. A Branch-and-Cut Approach

We next outline the polyhedral method of Fischetti and Toth [306]. Branch-and-cut methods for *ATSP* with sideconstraints have been proposedrecently by Ascheuer [43], Ascheuer, Jünger and Reinelt [46], and Ascheuer, Fischetti and Grötschel [44, 45], among others. The Fischetti-Toth method is based on model (1)–(6), and exploits additional classes of facet-inducing inequalities for the *ATSP* polytope *P* that proved to be of crucial importance for the solution of some real-world instances. For each class, we will address the associated *separation* problem (in its optimization version), defined as follows: Given a point $x^* \geq 0$ satisfying the degree equations, and a family $\mathcal{F}$ of *ATSP* inequalities, find a most violated member of $\mathcal{F}$, i.e., an inequality $\alpha x \leq \alpha_0$ belonging to $\mathcal{F}$ and maximizing the degree of violation $\alpha x^* - \alpha_0$. The reader is referred to Chapter 3 of the present book for a polyhedral analysis of the *ATSP* polytope, and to Chapter 2 for the design of branch-and-cut methods for the symmetric *TSP*.

To simplify notation, for any $f : A \rightarrow \mathbb{R}$ and $S_1, S_2 \subseteq V$, we write $f(S_1, S_2)$ for $\sum_{i \in S_1} \sum_{j \in S_2} f_{ij}$; moreover, we write $f(i, S_2)$ or $f(S_1, i)$ whenever $S_1 = \{i\}$ or $S_2 = \{i\}$, respectively.

4.1. Separation of Symmetric Inequalities

An *ATSP* inequality $\alpha x \leq \alpha_0$ is called *symmetric* when $\alpha_{ij} = \alpha_{ji}$ for all $(i,j) \in A$. Symmetric inequalities can be thought of as derived from valid inequalities for the *Symmetric Traveling Salesman Problem (STSP),* defined as the problem of finding a minimum-cost Hamiltonian cycle in a given undirected graph $G_E = (V, E)$. Indeed, let $y_e = 1$ if edge $e \in E$ belongs to the optimal *STSP* solution; $y_e = 0$ otherwise. Every inequality $\sum_{e \in E} \alpha_e y_e \leq \alpha_0$ for *STSP* can be transformed into a valid *ATSP* inequality by simply replacing y_e by $x_{ij} + x_{ji}$ for all edges $e = (i,j) \in E$. This produces the symmetric inequality $\alpha x \leq \alpha_0$, where $\alpha_{ij} = \alpha_{ji} = \alpha_{(i,j)}$ for all $i, j \in V$, $i \neq j$. Conversely, every symmetric *ATSP* inequality $\alpha x \leq \alpha_0$ corresponds to the valid *STSP* inequality $\sum_{(i,j) \in E} \alpha_{ij} y_{(i,j)} \leq \alpha_0$

The above correspondence implies that every separation algorithm for *STSP* can be used, as a "black box", for *ATSP* as well. To this end, given the *ATSP* (fractional) point x^* one first defines the undirected counterpart y^* of x^* by means of the transformation

$$y_e^* := x_{ij}^* + x_{ji}^* \qquad \text{for all edges } e = (i,j) \in E$$

and then applies the *STSP* separation algorithm to y^*. On return, the detected most violated *STSP* inequality is transformed into its ATSP counterpart, both inequalities having the same degree of violation.

Several exact/heuristic separation algorithms for *STSP* have been proposed in recent years, all of which can be used for *ATSP;* see Chapter 2 of the present book for further details. In [306] only two such separation tools are used, namely:

i) the Padberg-Rinaldi [646] exact algorithm for SECs; and

ii) the simplest heuristic scheme for comb (actually, 2-matching) constraints in which the components of the graph induced by the edges $e \in E$ with fractional y_e^* are considered as potential handles of the comb.

4.2. Separation of D_k^+ and D_k^- Inequalities

The following D_k^+ *inequalities* have been proposed by Grötschel and Padberg [405]:

$$x_{i_1 i_k} + \sum_{h=2}^{k} x_{i_h i_{h-1}} + 2\sum_{h=2}^{k-1} x_{i_1 i_h} + \sum_{h=3}^{k-1} x(\{i_2, \ldots, i_{h-1}\}, i_h) \le k-1 \quad (10)$$

where $(i_1, \ldots, i_k)$ is any sequence of $k \in \{3, \ldots, n-1\}$ distinct vertices, D_k^+ inequalities are facet-inducing for the *ATSP* polytope [295], and are obtained by lifting the cycle inequality $\sum_{(i,j)\in C} x_{ij} \le k-1$ associated with the subtour $C := \{(i_1, i_k), (i_k, i_{k-1}), \ldots, (i_2, i_1)\}$. Notice that the vertex indices along C are different from those used in the original Grötschel-Padberg definition [405], so as to allow for a simplified description of the forthcoming separation procedure.

As a slight extension of the original definition, we allow for $k = 1, 2$ in the sequel, in which cases (10) degenerates into the valid constraints $x_{i_1 i_1} \le 0$ and $x_{i_1 i_2} + x_{i_2 i_1} \le 1$, respectively.

The separation problem for the class of D_k^+ inequalities calls for a vertex sequence $(i_1, \ldots, i_k)$, $1 \le k \le n-1$, for which the degree of violation

$$\phi(i_1, \ldots, i_k) := x^*_{i_1 i_k} + \sum_{h=2}^{k} x^*_{i_h i_{h-1}} + 2\sum_{h=2}^{k-1} x^*_{i_1 i_h}$$

$$+ \sum_{h=3}^{k-1} x^*(\{i_2, \ldots, i_{h-1}\}, i_h) - k + 1 \quad (11)$$

is as large as possible. This is itself a combinatorial optimization problem that can be solved by the following simple implicit enumeration scheme.

We start with an empty node sequence. Then, iteratively, we extend the current sequence in any possible way and evaluate the degree of violation of the corresponding D_k^+ inequality. The process can be visualized by means of a branch-decision tree. The root node (level 0) of the tree represents the empty sequence. Each node at level k $(1 \le k \le n-1)$ corresponds to a sequence of the type $(i_1, \ldots, i_k)$; when $k \le n-2$, each such node generates $n-k$ descending nodes, one for each possible extended sequence $(i_1, \ldots, i_k, i_{k+1})$. Exhaustive enumeration of all nodes of the tree is clearly impractical, even for small values of n. On the other hand, a very large number of these nodes can be pruned (along with the associated subtrees) by means of the following simple upper bound computation. Let $(i_1, \ldots, i_k)$ be the sequence associated with the current branching node, say μ, and let ϕ_{max} denote the maximum degree of violation so far found during the enumeration. Consider any potential descendent node of μ, associated with a sequence of the type $(i_1, \ldots, i_k, i_{k+1}, \ldots, i_m)$. Then, directly from definition (11) one has

$$\phi(i_1, \ldots, i_k, i_{k+1}, \ldots, i_m) \leq \pi(x^*; i_1, \ldots, i_k) + x^*_{i_{k+1} i_k} + [x^*(i_1, V) - 1]$$
$$+ \sum_{h=k+1}^{m-1} [x^*(V, i_h) - 1] = \pi(i_1, \ldots, i_k) + x^*_{i_{k+1} i_k} \tag{12}$$

where we have defined

$$\pi(i_1, \ldots, i_k) := \sum_{h=2}^{k} x^*_{i_h i_{h-1}} + \sum_{h=2}^{k} x^*(\{i_1, \ldots, i_{h-1}\}, i_h) - k + 1$$

Observe that $\pi(i_1, \ldots, i_k)$ cannot exceed the degree of violation of the SEC associated with $S := \{i_1, \ldots, i_k\}$; hence one has $\pi(i_1, \ldots, i_k) \leq 0$ whenever all SECs are satisfied by x^*

According to (12), the only descending nodes of μ that need to be generated are those associated with a sequence $(i_1, \ldots, i_k, i_{k+1})$ such that

$$x^*_{i_k i_{k+1}} > \phi_{max} - \pi(i_1, \ldots, i_k) \tag{13}$$

Notice that both quantities $\phi(i_1, \ldots, i_k)$ and $\pi(i_1, \ldots, i_k)$ can parametrically be computed along the branching tree as:

$$\phi(i_1, \ldots, i_k) = \phi(i_1, \ldots, i_{k-1}) + x^*_{i_1 i_k} + x^*_{i_k i_{k-1}} + x^*(\{i_1, \ldots, i_{k-2}\}, i_{k-1}) - 1$$

and

$$\pi(i_1, \ldots, i_k) = \pi(i_1, \ldots, i_{k-1}) + x^*_{i_k i_{k-1}} + x^*(\{i_1, \ldots, i_{k-1}\}, i_k) - 1$$

where $\phi(i_1) := \pi(i_1) := 0$ for all singleton sequences (i_1)

Restriction (13) is very effective in practice, and dramatically reduces the number of nodes typically generated in the enumeration. Nevertheless, in some cases one may be interested in further reducing the computing time spent in the procedure. To this end, before running the above-described exact enumeration, one can try a "truncated" version of it in which each node at level $k \geq 2$ generates at most one descending node, namely the one associated with the sequence $(i_1, \ldots, i_k, i_{k+1})$, if any, where $x^*_{i_{k+1} i_k} > 0$ and $x^*_{i_1 i_{k+1}} + x^*_{i_{k+1} i_k}$ is as large as possible.

The performance of the overall branch-and-cut algorithm is generally improved if one generates, at each round of separation, a number of violated cuts (rather than the most violated one) for each family. In [306], the most violated D_k^+ inequality associated with a node sequence starting with i_1 is generated for each $i_1 \in V$. This is obtained by searching the decision tree in a depth-first manner, and resetting to zero the value ϕ_{max} of the incumbent best sequence whenever one backtracks to a node at level 1.

We conclude this section by addressing the following D_k^- *inequalities*:

$$x_{i_k i_1} + \sum_{h=2}^{k} x_{i_{h-1} i_h} + 2\sum_{h=2}^{k-1} x_{i_h i_1} + \sum_{h=3}^{k-1} x(i_h, \{i_2, \ldots, i_{h-1}\}) \le k-1 \quad (14)$$

where $(i_1, \ldots, i_k)$ is any sequence of $k \in \{3, \ldots, n-1\}$ distinct nodes, D_k^- inequalities are valid [405] and facet-inducing [295] for *P*; they can be obtained by lifting the cycle inequality $\sum_{(i,j)\in C} x_{ij} \le k-1$ associated with the directed cycle $C := \{(i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, i_1)\}$.

D_k^- inequalities can be thought of as derived from D_k^+ inequalities by swapping the coefficient of the two arcs (i,j) and (j,i) for all $i, j \in V$, $i < j$. This is a perfectly general operation, called *transposition* in [405], that works as follows.

For every $\alpha \in \mathbb{R}^A$, let $\alpha^T \in \mathbb{R}^A$ be defined by: $\alpha_{ij}^T := \alpha_{ji}$ for all $(i,j) \in A$. Clearly, inequality $\alpha x \le \alpha_0$ is valid (or facet-inducing) for the *ATSP* polytope *P* if and only if its transposed version, $\alpha^T x \le \alpha_0$, is. This follows from the obvious fact that $\alpha^T x = \alpha x^T$, where $x \in P$ if and only if $x^T \in P$. Moreover, every separation procedure for $\alpha x \le \alpha_0$ can also be used, as a black box, to deal with $\alpha^T x \le \alpha_0$. To this end one gives the transposed point $(x^*)^T$ (instead of x^*) on input to the procedure, and then transposes the returned inequality.

The above considerations show that both the heuristic and exact separation algorithms designed for D_k^+ inequalities can be used for D_k^- inequalities as well.

4.3. Separation of Odd CAT Inequalities

The following class of inequalities has been proposed by Balas [62]. Two distinct arcs (i,j) and (u,v) are called *incompatible* if $i = u$, or $j = v$, or $i = v$ and $j = u$; *compatible* otherwise. A *Closed Alternating Trail* (CAT, for short) is a sequence $T = \{a_1, \ldots, a_t\}$ of t distinct arcs such that, for $k = 1, \ldots, t$, arc a_k is incompatible with arcs a_{k-1} and a_{k+1}, and compatible with all other arcs in *T* (with $a_0 := a_t$ and $a_{t+1} := a_1$). Let $\delta^+(v)$ and $\delta^-(v)$ denote the set of the arcs of *G* leaving and entering any vertex $v \in V$, respectively. Given a CAT *T*, a node v is called a *source* if $|\delta^+(v) \cap T| = 2$, whereas it is called a *sink* if $|\delta^-(v) \cap T| = 2$. Notice that a node can play both source and sink roles. Let *Q* be the set of the arcs $(i,j) \in A \setminus T$ such that *i* is a source and *j* is a sink node.

For any CAT of odd length t, the following *odd CAT inequality*

$$\sum_{(i,j)\in T\cup Q} x_{ij} \le \frac{|T|-1}{2} \tag{15}$$

is valid and facet-defining (except in two pathological cases arising for $n \le 6$) for the *ATSP* polytope [62].

We next describe a heuristic separation algorithm for the family of odd CAT inequalities. This algorithm is based on the known fact that odd CAT inequalities correspond to odd cycles on an auxiliary "incompatibility" graph [62]. Also, the separation algorithm can be viewed as a specialized version of a scheme proposed by Caprara and Fischetti [159] for the separation of a subclass of Chvátal-Gomory cuts for general integer programming problems.

Given the point x^*, we set-up an edge-weighted undirected graph $\tilde{G} = (\tilde{N}, \tilde{E})$ having a node ν_a for each arc $a \in A$ with $x^*_a > 0$, and an edge $e = (\nu_a, \nu_b)$ for each pair a, b of incompatible arcs, whose weight is defined as $w_e := 1 - (x^*_a + x^*_b)$. We assume that x^* satisfies all degree equations as well as all trivial SECs of the form $x_{ij} + x_{ji} \le 1$; this implies $w_e \ge 0$ for all $e \in \tilde{E}$.

Let $\tilde{\delta}(v)$ contain the edges in $\tilde{E}$ incident with a given node $v \in \tilde{N}$. A *cycle* $\tilde{C}$ of $\tilde{G}$ is an edge subset of $\tilde{E}$ inducing a connected subdigraph of $\tilde{G}$, and such that $|\tilde{C} \cap \tilde{\delta}(v)|$ is even for all $v \in \tilde{N}$. Cycle $\tilde{C}$ is called (i) *odd* if $|\tilde{C}|$ is odd; (ii) *simple* if $|\tilde{C} \cap \tilde{\delta}(v)| \in \{0, 2\}$ for all $v \in \tilde{N}$; and (iii) *chordless* if the subdigraph of $\tilde{G}$ induced by the nodes covered by $\tilde{C}$ has no other edges than those in $\tilde{C}$

By construction, every simple and chordless odd cycle $\tilde{C}$ in $\tilde{G}$ corresponds to an odd CAT T, where $a \in T$ if and only if ν_a is covered by $\tilde{C}$ In addition, the total weight of $\tilde{C}$ is

$$w(\tilde{C}) := \sum_{e\in\tilde{C}} w_e = \sum_{(\nu_a,\nu_b)\in\tilde{C}} (1 - x^*_a - x^*_b) = |T| - 2\sum_{a\in T} x^*_a$$

hence $(1 - w(\tilde{C}))/2$ gives a lower bound on the degree of violation of the corresponding CAT inequality, computed as

$$\phi(T) := (2\sum_{a\in T\cup Q} x^*_a - |T| + 1)/2$$

The heuristic separation algorithm used in [306] computes, for each $e \in \tilde{E}$, a minimum-weight odd cycle $\tilde{C}_e$ that uses edge e. If $\tilde{C}_e$ happens to be simple and chordless, then it corresponds to an odd CAT, say T. If, in addition, the lower bound $(1 - w(\tilde{C}_e))/2$ exceeds a given threshold $\theta =$

–1/2, then the corresponding inequality is hopefully violated; hence one evaluates its actual degree of violation, $\phi(T)$, and stores the inequality if $\phi(T) > 0$. In order to avoid detecting twice the same inequality, edge e is removed from $\tilde{G}$ after the computation of each $\tilde{C}_e$.

In order to increase the chances of finding odd cycles that are simple and chordless, all edge weights can be made strictly positive by adding to them a small positive value $\epsilon := 0.001$. This guarantees that ties are broken in favor of inclusion-minimal sets $\tilde{C}_e$. Notice, however, that a generic minimum-weight odd cycle $\tilde{C}_e$ does not need to be neither simple nor chordless even in this case, due to the fact that one imposes $e \in \tilde{C}_e$. For example, $\tilde{C}_e$ may decompose into 2 simple cycles, say $\tilde{C}_e^1$ and $\tilde{C}_e^2$, where $\tilde{C}_e^1$ is of even cardinality and goes through edge e., and $\tilde{C}_e^2$ is of odd cardinality and overlaps $\tilde{C}_e^1$ in a node.

The key point of the algorithm is the computation in $\tilde{G}$ of a minimum-weight odd cycle going through a given edge. Assuming that the edge weights are all nonnegative, this problem is known to be polynomially solvable as it can be transformed into a shortest path problem; see Gerards and Schrijver [357]. To this end one constructs an auxiliary bipartite undirected graph $G_B = (N'_B \cup N''_B, E_B)$ obtained from $\tilde{G}$ as follows. For each ν in $\tilde{G}$ there are two nodes in G_B, say ν' and ν''. For each edge $e = (\nu_1, \nu_2)$ of $\tilde{G}$ there are two edges in G_b, namely edge (ν'_1, ν''_2) and edge (ν'_2, ν''_1), both having weight w_e. By construction, every minimum-weight odd cycle $\tilde{C}_e$ of $\tilde{G}$ going through edge $e = (\nu_1, \nu_2)$ corresponds in G_B to a shortest path from ν'_1 to ν'_2, plus the edge (ν'_2, ν''_1). Hence, the computation of all $\tilde{C}_e$'s can be performed efficiently by computing, for each ν'_1, the shortest path from ν'_1 to all other nodes in N'_B

4.4. Clique Lifting and Shrinking

Clique lifting can be described as follows, see Balas and Fischetti [73] for details. Let $P(G')$ denote the *ATSP* polytope associated with a given complete digraph $G' = (V', A')$. Given a valid inequality $\beta y \leq \beta_0$ for $P(G')$, we define

$$\beta_{hh} := \max\{\beta_{ih} + \beta_{hj} - \beta_{ij} : i, j \in V' \setminus \{h\}, i \neq j\} \quad \text{for all } h \in V'$$

and construct an enlarged complete digraph $G = (V, A)$ obtained from G' by replacing each node $h \in V'$ by a clique S_h containing at least one node (hence, $|V| = \sum_{h \in V'} |S_h| \geq |V'|$). In other words $(S_1, \ldots, S_{|V'|})$ is a proper partition of V, in which the h-th set corresponds to the h-th node in V'.

For all $v \in V$, let $v \in S_{h(v)}$. We define a new *clique lifted* inequality for $P(G)$, say $\alpha x \leq \alpha_0$, where $\alpha_0 := \beta_0 + \sum_{h \in V'} \beta_{hh}(|S_h| - 1)$ and

$\alpha_{ij} := \beta_{h(i)h(j)}$ for each $(i,j) \in A$. It is shown in [73] that the new inequality is always valid for $P(G)$; in addition, if the starting inequality $\beta x \le \beta_0$ defines a facet of $P(G')$, then $\alpha x \le \alpha_0$ is guaranteed to be facet-inducing for $P(G)$.

Clique lifting is a powerful theoretical tool for extending known classes of inequalities. Also, it has important applications in the design of separation algorithms in that it allows one to simplify the separation problem through the following *shrinking* procedure [646].

Let $S \subset V$, $2 \le |S| \le n-2$, be a vertex subset saturated by x^*, in the sense that $x^*(S,S) = |S| - 1$, and suppose S is shrunk into a single node, say σ, and x^* is updated accordingly. Let $G' = (V', A')$ denote the shrunken digraph, where $V' := V \setminus S \cup \{\sigma\}$, and let y^* be the shrunken counterpart of x^*. Every valid inequality $\beta y \le \beta_0$ for $P(G')$ that is violated by y^* corresponds in G to a violated inequality, say $\alpha x \le \alpha_0$, obtained through clique lifting by replacing back σ with the original set S. As observed by Padberg and Rinaldi [647], however, this shrinking operation can affect the possibility of detecting violated cuts on G', as it may produce a point y^* belonging to $P(G')$ even when $x^* \notin P(G)$

For instance, let $n := 4$ and $x^*_{ij} := 1/2$ if $(i,j) \in \{(1,2),(1,4),(2,1),$ (2,3), (3,1), (3,4), (4,2), (4,3)$\}$; $x^*_{ij} = 0$ otherwise. One readily checks that $x^* \notin P(G)$, as x^* violates, e.g., the D_3^+ inequality $x_{12} + x_{23} + x_{31} + 2x_{21} \le 2$. On the other hand, shrinking the saturated set $S := \{1,2\}$ produces a digraph G' with vertex set $V' := \{\sigma, 3, 4\}$, and a point y^* with $y^*_{ij} = 1/2$ for all $(i,j) \in A'$. But then y^* is the convex combination of the characteristic vectors of the two tours $(\sigma, 3, 4)$ and $(\sigma, 4, 3)$, hence y^* cannot be cut off by any linear inequality as it belongs to $P(G')$.

The above example shows that shrinking has to be applied with some care. There are however simple conditions on the choice of S that guarantee $y^* \notin P(G')$, provided $x^* \notin P(G)$ as in the cases of interest for separation.

The simplest such condition concerns the shrinking of *1-arcs* (i.e., arcs (i,j) with $x^*_{ij} = 1$), and requires $S = \{i,j\}$ for a certain node pair i,j with $x^*_{ij} = 1$. To see the validity of the condition, assume by contradiction that $y^* \in P(G')$. This implies $y^* = \sum_{k=1}^{M} \lambda_k y^k$, where $y^1, \ldots, y^M$ are characteristic vectors of tours in G', and $\lambda_1, \ldots, \lambda_M$ are nonnegative multipliers with $\sum_{k=1}^{M} \lambda_k = 1$. For each $k \in \{1, \ldots, M\}$ we define x^k as the characteristic vector of the tour of G obtained from y^k by replacing node σ with the arc (i,j). Then, by construction, $x^* = \sum_{k=1}^{M} \lambda_k x^k$, which contradicts the assumption $x^* \notin P(G)$.

It is known that 1-edges cannot be shrunk for *STSP,* instead. In this respect *ATSP* behaves more nicely than *STSP,* in that the information associated with the orientation of the arcs allows for more powerful shrinkings. Here is a polyhedral interpretation of this behavior.

In the separation problem, we are given a point x^* which satisfies the valid inequality $x_{ij} \leq 1$ with equality, and we want to separate it from the *ATSP* polytope, *P*. The above discussion shows that this is possible if and only if x^* can be separated from $F := \{x \in P : x_{ij} = 1\}$, hence *F* can replace *P* insofar the separation of x^* is concerned. This property is perfectly general, and applies to any nonempty face *F* of any polytope *P*. Indeed, let $F := \{x \in P : \beta x = \beta_0\} \neq \emptyset$ be the face of *P* induced by any valid inequality $\beta x \leq \beta_0$ for *P*, and assume that $\beta x^* = \beta_0$. If x^* cannot be separated from *F*, then $x^* \in F \subseteq P$, hence it cannot be separated from *P* too. Conversely, suppose there exists a valid inequality $\alpha x \leq \alpha_0$ for *F* which is violated by x^* by the amount $\delta^* := \alpha x^* - \alpha_0 > 0$. Then for a sufficiently large value *M* the "lifted" inequality $(\alpha + M\beta)x \leq \alpha_0 + M\beta_0$ is valid for *P*, but violated by x^* by the amount $(\alpha + M\beta)x^* - (\alpha_0 + M\beta_0) = \alpha x^* - \alpha_0 = \delta^* > 0$

The different behavior between the asymmetric and symmetric TSP polytopes then has its roots in the basic property that fixing $x_{ij} = 1$ for some *ATSP* arc (i, j) yields again an *ATSP* instance – obtained by contracting (i, j) into a single vertex – whereas the same construction does not work when fixing $y_e = 1$ for some *STSP* undirected edge e. In polyhedral terms, this means that the face *F* of the *ATSP* polytope induced by $x_{ij} \leq 1$ is in 1-1 correspondence with the ATSP polytope on $n-1$ nodes. Hence, in the asymmetric case, the face *F* can be interpreted again as an *ATSP* polytope on a shrunken graph, whereas for *STSP* a similar interpretation is not possible.

In [306] 1-arc shrinking is applied iteratively, so as to replace each path of 1-arcs by a single node. As a result of this pre-processing on x^*. all the nonzero variables are fractional. Notice that a similar result cannot be obtained for the symmetric TSP, where each 1-edge chain can be replaced by a single 1-edge, but not by a single node.

More sophisticated shrinking procedures have not been used in [306]. The above discussion suggests however other cases in which a saturated set *S* can be shrunk. For instance, suppose there exist 3 distinct nodes i, j and k such that $x^*_{ij} + x^*_{ji} = 1$ and $x^*_{jk} + x^*_{kj} = 1$, i.e., $x^*_{ij} = x^*_{jk} = \mu$ and $x^*_{kj} = x^*_{ji} = 1 - \mu$ for some $0 \leq \mu \leq 1$. We claim, that in this case the saturated set $S = \{i, j\}$ can be shrunk. Indeed, the given point x^* satisfies the valid *ATSP* inequality $\beta x := x_{ij} + x_{ji} + x_{jk} + x_{kj} \leq \beta_0 := 2$ with equality, hence from the above discussion one can replace *P* with its face $F := \{x \in P : \beta x = \beta_0\}$. Now, every extreme point of *F* corresponds to a tour using either the path $\{(i, j), (j, k)\}$ or the path $\{(k, j), (j, i)\}$. This property induces a 1-1 correspondence between the extreme points of F and those of the *ATSP* polytope in which i and j have been shrunk into a single node.

4.5. Pricing with Degeneracy

Pricing is an important ingredient of branch-and-cut codes, in that it allows one to effectively handle LP problems involving a huge number of columns. Let

$$z := \min\{cx : Mx \equiv b, x \geq 0\} \tag{16}$$

be the LP problem to be solved. M is an $m \times |A|$ matrix whose columns are indexed by the arcs $(i,j) \in A$. The first $2n-1$ rows of M correspond to the degree equations (2)-(3) (with the redundant constraint $x(1,V) = 1$ omitted), whereas the remaining rows, if any, correspond to some of the cuts generated through separation. Notation "$\equiv$" stands for "=" for the first $2n-1$ rows of M, and "$\leq$" for the remaining rows. Let M_{ij}^h denote the entry of M indexed by row h and column (i,j).

In order to keep the size of the LP as small as possible, the following *pricing* scheme is commonly used. We determine a (small) *core* set of arcs, say $\tilde{A}$, and decide to temporarily fix $x_{ij} = 0$ for all $(i,j) \in A \setminus \tilde{A}$. We then solve the restricted LP problem

$$\tilde{z} := \min\{\tilde{c}\tilde{x} : \tilde{M}\tilde{x} \equiv b, \tilde{x} \geq 0\} \tag{17}$$

where $\tilde{c}$, $\tilde{x}$, and $\tilde{M}$ are obtained from c, x, and M, respectively, by removing all entries indexed by $A \setminus \tilde{A}$.

Assume problem (17) is feasible, and let $\tilde{x}^*$ and $\tilde{u}^*$ be the optimal primal and dual basic solutions found, respectively. Clearly, $\tilde{z} \geq z$. We are interested in easily-checkable conditions that guarantee $\tilde{z} = z$, thus proving that $\tilde{x}^*$ (with $\tilde{x}_{ij}^* := 0$ for all $(i,j) \in A \setminus \tilde{A}$) is an optimal basic solution to (16), and hence that its value $\tilde{z}$ is a valid lower bound on $v(ATSP)$. To this end we compute the LP reduced costs associated with $\tilde{u}^*$, namely

$$\bar{c}_{ij} := c_{ij} - \sum_{h=1}^{m} M_{ij}^h \tilde{u}_h^* \quad \text{for } (i,j) \in A$$

and check whether $\bar{c}_{ij} \geq 0$ for all $(i,j) \in A$. If this is indeed the case, then $\tilde{z} = z$ and we are done. Otherwise, the current core set $\tilde{A}$ is enlarged by adding (some of) the arcs with negative reduced cost, and the whole procedure is iterated. This iterative solution of (17), followed by the possible updating of $\tilde{A}$, is generally referred to as the *pricing loop*.

According to common computational experience, the first iterations of the pricing loop tend to add a very large number of new columns to the LP even when $\tilde{z} = z$, due to the typically high primal degeneracy of (17).

As an illustration of this behavior, consider the situation arising when the first LP is solved at the root node of the branching tree. In this case (16) contains the $2n-1$ degree equations only, hence its optimal solution, x^*, can be computed efficiently through any *AP* code. Suppose we now initialize the core set $\tilde{A}$ to contain the n arcs chosen in the optimal *AP* solution. In order to have an LP basis, we add $n-1$ additional arcs to $\tilde{A}$, chosen so as to determine an $(2n-1) \times (2n-1)$ nonsingular matrix $\tilde{M}$. By construction, problem (17) has a unique feasible solution – namely, the characteristic vector of the optimal *AP* solution found – hence we know that $\tilde{z} = z$ holds in this case. However, depending on the possibly "wrong" choice of the last $n-1$ arcs in the core set, the solution $\tilde{u}^*$ can be dual infeasible for (16), i.e., a usually very large number of reduced costs $\overline{c}_{ij}$ are negative. Iterating the pricing procedure produces a similar behavior, and a long sequence of pricings is typically required before all arcs price-out correctly.

The above example shows that checking the reduced-cost signs can lead to an overweak sufficient condition for proving $\tilde{z} = z$. The standard way to cope with this weakness consists in a more careful initialization of the core set, e.g., by taking the 15 smallest-cost arcs leaving each node.

We next describe a different technique, called *AP pricing* in [306], in which the pricing condition is strengthened by exploiting the fact that any feasible solution to (16) cannot select the arcs with negative reduced cost in an arbitrary way, as the degree equations —among other constraints— have to be fulfilled. The technique is related to the so-called *Lagrangian pricing* introduced independently by Löbel [566] as a powerful method for solving large-scale vehicle scheduling problems.

Let us consider the dual solution $\tilde{u}^*$ to (17) as a vector of Lagrangian multipliers, and the LP reduced costs $\overline{c}_{ij}$ as the corresponding Lagrangian costs. In this view, standard pricing consists of solving the following trivial relaxation of (16):

$$LB_1 := \min_{x \geq 0}[cx + \tilde{u}^*(b - Mx)] = \tilde{u}^* b + \min_{x \geq 0} \overline{c}x \tag{18}$$

where $\tilde{u}^* b = \tilde{z}$ by LP duality. Therefore one has $\tilde{z} + \min_{x \geq 0} \overline{c}x \leq z \leq \tilde{z}$, from which $\tilde{z} = z$ in case $\min_{x \geq 0} \overline{c}x = 0$, i.e., $\overline{c}_{ij} \geq 0$ for all i, j. The strengthening then consists in replacing condition $x \geq 0$ in (18) by

$$x \in F(AP) := \{x \in \{0,1\}^A : x(i, V) = x(V, i) = 1 \text{ for all } i \in V\}$$

In this way we compute an improved lower bound on z, namely

$$LB_2 := \tilde{u}^* b + \min_{x \in F(AP)} \overline{c}x = \tilde{z} + \Delta_{AP}$$

where $\Delta_{AP} := \min_{x \in F(AP)} \overline{c}x$ is computed efficiently by solving the *AP* on the Lagrangian costs $\overline{c}_{ij}$. As before, $\tilde{z}+\Delta_{AP} \leq z \leq \tilde{z}$, hence $\Delta_{AP} = 0$ implies $\tilde{z} = z$. When $\Delta_{AP} < 0$, instead, one has to iterate the procedure, after having added to the core set $\tilde{A}$ the arcs in $A \setminus \tilde{A}$ that are selected in the optimal *AP* solution found.

The new approach has two main advantages, namely: (1) an improved check for proving $\tilde{z} = z$; and (2) a better rule to select the arcs to be added to the core arc set. Moreover, LB_2 always gives a lower bound on z (and hence on $v(ATSP)$), which can in some cases succeed in fathoming the current branching node even when $\Delta_{AP} < 0$. Finally, the nonnegative *AP* reduced cost vector $\overline{\overline{c}}$ available after solving $\min_{x \in F(AP)} \overline{c}x$ can be used for fixing $x_{ij} = 0$ for all $(i, j) \in A$ such that $LB_2 + \overline{\overline{c}}_{ij}$ is at least as large as the value of the best known *ATSP* solution.

The new pricing scheme can be adapted to other problems having an easily-solvable relaxation. For example, in Fischetti and Vigo [307] the approach is applied to the resource constrained arborescence problem, the relaxation used for pricing being in this case the min-sum arborescence problem. Unlike *AP,* the latter problem involves exponentially many constraints, hence the pricing scheme can in some cases also detect violated cuts that are not present in the current LP, chosen from among those implicitly used during the solution of the relaxation. In other words, variable-pricing also produces, as a by-product, a heuristic "pricing" of someexponential classes of cuts, i.e., a separation tool.

AP pricing requires the computation of all reduced costs $\overline{c}_{ij}$, e.g., through the following "row-by-row" scheme. We initialize $\overline{c}_{ij} := c_{ij}$ for all $(i, j) \in A$ and then consider, in sequence, the rows h of *M* with $\tilde{u}_h^* \neq 0$. For each such row h, we determine the set $A_h := \{(i, j) \in A : M_{ij}^h \neq 0\}$, and update $\overline{c}_{ij} := \overline{c}_{ij} - \tilde{u}_h^* M_{ij}^h$ for $(i, j) \in A_h$. Because of the simple combinatorial structure of most cuts, the (implicit) construction of A_h can typically be carried out in $O(|A_h|)$ time, hence the overall time spent for computing all reduced costs is $O(n^2)$ for the initialization, plus $O(\sum_{h=1}^{m} |A_h|)$ for the actual reduced-cost computation (notice that this latter term is linear in the number of nonzero entries in *M*).

A drawback of the *AP* pricing is the extra computing time spent for the *AP* solution, which can however be reduced considerably through the following strategy [306]. After each LP solution we compute the reduced costs $\overline{c}_{ij}$ and determine the cardinality μ of $A^- := \{(i, j) \in A : \overline{c}_{ij} < 0\}$. If $\mu = 0$, we exit the pricing loop. If $0 < \mu \leq n/10$, we use standard pricing, i.e., we update $\tilde{A} := \tilde{A} \cup A^-$ and repeat. If $\mu > n/10$, instead, we resort to *AP* pricing, and solve the *AP* problem on the reduced costs. We also determine (through a technique described, e.g., in [306]) the arc set *Q* containing the $2n - 1$ arcs defining an optimal LP basis for this *AP* problem, and compute the corresponding nonnegative reduced costs $\bar{\bar{c}}_{ij}$ for all $(i,j) \in A$. We then remove from Q all the arcs already in $\tilde{A}$, and enlarge Q by iteratively adding arcs in $(i,j) \in A \setminus (\tilde{A} \cup Q)$ with $\bar{\bar{c}}_{ij} = 0$, until no such arc exists, or $|Q| \geq 50 + n/3$. In this way Q contains a significant number of arcsthat are likely to be selected in (16). We finally update $\tilde{A} := \tilde{A} \cup Q$ (even in case $\Delta_{AP} = 0$), and repeat (if $\Delta_{AP} < 0$) or exit (if $\Delta_{AP} = 0$) the pricing loop.

4.6. The Overall Algorithm

The algorithm is a lowest-first branch-and-cut procedure. At each node of the branching tree, the LP relaxation is initialized by taking all the constraints present in the last LP solved at the father node (for the root node, only the degree equations are taken). As to variables, one retrieves from a scratch file the optimal basis associated with the last LP solved at the father node, and initializes the core variable set. $\tilde{A}$, by taking all the arcs belonging to this basis (for the root node, $\tilde{A}$ contains the $2n-1$ variables in the optimal *AP* basis found by solving *AP* on the original costs c_{ij}). In addition, $\tilde{A}$ contains all the arcs of the best known *ATSP* solution. Starting with the above advanced basis, one iteratively solves the current LP, applies the *AP* pricing (and variable fixing) procedure described in Section 4.5, and repeats if needed. Observe that the pricing/fixing procedure is applied after each LP solution.

On exit of the pricing loop (case $\Delta_{AP} = 0$), the cuts whose associated slack exceeds 0.01 are removed from the current LP (unless the number of these cuts is less than 10), and the LP basis is updated accordingly. Moreover, separation algorithms are applied to find, if any, facet-defining *ATSP* inequalities that cut off the current LP optimal solution, say x^*. As a heuristic rule, the violated cuts with degree of violation less than 0.1 (0.01 for SECs) are skipped, and the separation phase is interrupted as soon as 20 + $\lfloor n/5 \rfloor$ violated cuts are found.

One first checks for violation the cuts generated during the processing of the current or previous nodes, all of which are stored in a global data-structure called the constraint *pool.* If some of these cuts are indeed violated by x^*, the separation phase ends. Otherwise, the Padberg-Rinaldi [646] MINCUT algorithm for SEC separation is applied, and the separation phase is interrupted if violated SECs are found. When this is not the case, one shrinks the 1-arc paths of x^* (as described in Section 4.4), and applies the separation algorithms for comb (Section 4.1), D_k^+ and D_k^- (Section 4.2), and odd CAT (Section 4.3) inequalities. In order to avoid finding equivalent inequalities, D_3^- inequalities (which are the same as D_3^+ inequalities), are never separated, and odd CAT separation is skipped when a violated comb is found (as the class of comb and odd CAT inequalities overlap). When violated cuts are found, one adds them to the current LP, and repeats.

When separation fails and x^* is integer, the current best *ATSP* solution is updated, and a backtracking step occurs. If x^* is fractional, instead, the current LP basis is saved in a file, and one branches on the variable x_{ij} with $0 < x^*_{ij} < 1$ that maximizes the score $\sigma(i,j) := c_{ij} \cdot \min\{x^*_{ij}, 1 - x^*_{ij}\}$. As a heuristic rule, a large priority is given to the variables with $0.4 \leq x^*_{ij} \leq 0.6$ (if any), so as to produce a significant change in both descending nodes.

As a heuristic tailing-off rule, one also branches when the current x^* is fractional and the lower bound did not increase in the last 5 (10 for the root node) LP/pricing/separation iterations.

A simple heuristic algorithm is used to hopefully update the current best optimal *ATSP* solution. The algorithm is based on the information associated with the current LP, and consists of a complete enumeration of the Hamiltonian directed cycles in the support graph of x^*, defined as $G^* := (V, \{(i,j) \in A : x^*_{ij} > 0\})$. To this end Martello's [585] implicit enumeration algorithm HC is used, with at most $100 + 10n$ backtracking steps allowed. As G^* is typically very sparse, this upper bound on the number of backtrackings is seldom attained, and HC almost always succeeds in completing the enumeration within a short computing time. The heuristic is applied whenever SEC separation fails, since in this case G^* is guaranteed to be strongly connected.

5. Computational Experiments

The algorithms described in the previous sections have been computationally tested on a large set of *ATSP* instances namely:

- 42 instances by Cirasella, Johnson, McGeoch and Zhang [200]; the instances in this set are randomly generated to simulate real-world applications arising in many different fields;
- 5 scheduling instances provided by Balas [67];
- 2 additional real-world instances (`ftv180` and `uk66`);
- 10 random instances whose integer costs are uniformly generated in range [1, 1000];
- all the 27 *ATSP* instances collected in TSPLIB [709].

For a detailed description of the first 42 instances the reader is referred to [200], while details for the ones in the TSPLIB can be found in the associated web page [709]. The 5 instances provided by Balas come from *Widget*, a generator of realistic instances from the chemical industry developed by Donald Miller. Instance `ftv180` represents pharmaceutical product delivery within Bologna down-town and is obtained from the TSPLIB instance `ftv170` by considering 10 additional vertices in the graph representing down-town Bologna. Finally, instance `uk66` is a real-world problem arising in routing applications and has been provided us by Kousgaard [518].

All instances have integer nonnegative costs, and are available, on request, from the authors. For each instance we report in Table 4.1 the name (Name), the size (n), the optimal (or best known) solution value (OptVal), and the source of the instance (source).

Three specific *ATSP* codes have been tested: (1) the *AP*-based branch-and-bound `CDT` code [163], as described in Section 2.1; (2) `FT-add` code, corresponding to the additive approach [304] described in Section 3; and (3) `FT-b&c` code, corresponding to the branch-and-cut algorithm [306] described in Section 4.

In addition, the branch-and-cut code `Concorde` by Applegate, Bixby, Chvátal and Cook [29] has been considered, as described in Chapter 2. This code is specific for the symmetric *TSP*, so symmetric instances have to be constructed through one of the following two transformations:

- the *3-node* transformation proposed by Karp [495]. A complete *undirected* graph with $3n$ vertices is obtained from the original complete *directed* one by adding two copies, $n+i$ and $2n+i$, of each vertex $i \in V$, and by (i) setting to 0 the cost of the edges $(i, n+i)$ and $(n+i, 2n+i)$ for each $i \in V$, (ii) setting to c_{ij} the cost of edge $(2n+i, j)$ $\forall i, j \in V$, and (iii) setting to $+\infty$ the costs of all the remaining edges;

- the *2-node* transformation proposed by Jonker and Volgenant [471] (see also Jünger, Reinelt and Rinaldi [474]). A complete *undirected* graph with $2n$ vertices is obtained from the original complete *directed* one by adding a copy, $n+i$, of each vertex $i \in V$, and by (i) setting to 0 the cost of the edge $(i, n+i)$ for each $i \in V$, (ii) setting to $c_{ij} + M$ the cost of edge $(n+i, j)$ $\forall i, j \in V$, where M is a sufficiently large positive value, and (iii) setting to $+\infty$ the costs of all the remaining edges. The transformation value nM has to be subtracted from the *STSP* optimal cost.

All tests have been executed on a Digital Alpha 533 MHz with 512 MB of RAM memory under the Unix Operating System, with Cplex 6.5.3 as LP solver. In all tables, we report the percentage gaps corresponding to the

Table 4.1. ATSP instances.

Name	n	OptVal	source	Name	n	OptVal	source
coin100.0	100	10360	[200]	balas84	84	199	[67]
coin100.1	100	10600	[200]	balas108	108	152	[67]
coin100.2	100	10980	[200]	balas120	120	286	[67]
coin100.3	100	10540	[200]	balas160	160	397	[67]
coin100.4	100	10440	[200]	balas200	200	403	[67]
coin316.10	316	32360	[200]	ftv180	181	2918	this Chapter
crane100.0	100	7777997	[200]	uk66	66	2791	[518]
crane100.1	100	7615069	[200]	ran1000.0	1000	7897	$c_{ij} \in [1, 1000]$
crane100.2	100	8062054	[200]	ran1000.1	1000	8003	$c_{ij} \in [1, 1000]$
crane100.3	100	7018782	[200]	ran1000.2	1000	7991	$c_{ij} \in [1, 1000]$
crane100.4	100	7786309	[200]	ran1000.3	1000	7956	$c_{ij} \in [1, 1000]$
crane316.10	316	13132907	[200]	ran1000.4	1000	8014	$c_{ij} \in [1, 1000]$
disk100.0	100	17471906	[200]	ran500.0	500	5507	$c_{ij} \in [1, 1000]$
disk100.1	100	18186198	[200]	ran500.1	500	5398	$c_{ij} \in [1, 1000]$
disk100.2	100	15568681	[200]	ran500.2	500	5394	$c_{ij} \in [1, 1000]$
disk100.3	100	18326394	[200]	ran500.3	500	5354	$c_{ij} \in [1, 1000]$
disk100.4	100	17862733	[200]	ran500.4	500	5337	$c_{ij} \in [1, 1000]$
disk316.10	316	28904480	[200]	br17	17	39	TSPLIB
rtilt100.0	100	9465148	[200]	ft53	53	6905	TSPLIB
rtilt100.1	100	9623330	[200]	ft70	70	38673	TSPLIB
rtilt100.2	100	9411004	[200]	ftv33	34	1286	TSPLIB
rtilt100.3	100	9584646	[200]	ftv35	36	1473	TSPLIB
rtilt100.4	100	10265172	[200]	ftv38	39	1530	TSPLIB
rtilt316.10	316	16738334	[200]	ftv44	45	1613	TSPLIB
shop100.0	100	143019	[200]	ftv47	48	1776	TSPLIB
shop100.1	100	147815	[200]	ftv55	56	1608	TSPLIB
shop100.2	100	148602	[200]	ftv64	65	1839	TSPLIB
shop100.3	100	148413	[200]	ftv70	71	1950	TSPLIB
shop100.4	100	144270	[200]	ftv90	91	1579	TSPLIB
shop316.10	316	427004	[200]	ftv100	101	1788	TSPLIB
stilt100.0	100	15214154	[200]	ftv110	111	1958	TSPLIB
stilt100.1	100	15543791	[200]	ftv120	121	2166	TSPLIB
stilt100.2	100	15199456	[200]	ftv130	131	2307	TSPLIB
stilt100.3	100	15499387	[200]	ftv140	141	2420	TSPLIB
stilt100.4	100	16303671	[200]	ftv150	151	2611	TSPLIB
stilt316.10	316	26715915*	[200]	ftv160	161	2683	TSPLIB
super100.0	100	785	[200]	ftv170	171	2755	TSPLIB
super100.1	100	780	[200]	kro124p	100	36230	TSPLIB
super100.2	100	780	[200]	p43	43	5620	TSPLIB
super100.3	100	776	[200]	rbg323	323	1326	TSPLIB
super100.4	100	809	[200]	rbg358	358	1163	TSPLIB
super316.10	316	2109	[200]	rbg403	403	2465	TSPLIB
				rbg443	443	2720	TSPLIB
				ry48p	48	14422	TSPLIB

*best know solution value.

lower bound at the root node (Root), the final lower bound (fLB), and the final upper bound (fUB), all computed with respect to the optimal (or best known) solution value. Moreover, the number of nodes of the search tree (Nodes), and the computing time in seconds (Time) are given.

5.1. Code Tuning

In this section we analyze variants of the above mentioned codes with the aim of determining, for each code, the best average behavior.

`CDT` **Code.** No specific modifications of this code have been implemented.

`FT-add` **Code.** The original implementation emphasized the lower bounding procedures. As in the `CDT code`, improved performances of the overall algorithm have been obtained by using the patching heuristic of Karp [498] (applied at each node), and the subtour merging operation described in Section 2.1.3. This leads to a reduction of both the number of branching nodes and the overall computing time; in particular, instances `balas84` and `ftv160` could not be solved by the original code within the time limit of 1,000 CPU seconds.

`FT-b&c` **Code.** A new branching criterion, called *Fractionality Persistency* (FP) has been tested. Roughly speaking, the FP criterion gives priority for branching to the variables that have been persistently fractional in the last LP optimal solutions determined at the current branching node. This general strategy has been proposed and computationally analyzed in [298]. Table 4.2 compares the original and modified codes on a relevant subset of instances when imposing a time limit of 10,000 CPU seconds. According to these results, the FP criterion tends to avoid pathological situations due to branching (two more instances solved to optimality), and leads to a more robust code.

`Concorde` **Code.** The code has been used with default parameters by setting the *random seed* parameter (“`-s 123`”) so as to be able to reproduce each run. Both *ATSP*-to-*STSP* transformations have been tested by imposing a time limit of 10,000 CPU seconds. For the *2-node* transformation, parameter M has been set to 1,000,000 (but for the instances with $n = 1{,}000$ and for instance `rtilt316.10`, for which we used $M = 100,000$ so as to avoid numerical problems). The comparison of the results obtained by `Concorde` by using the two transformations is given in Table 4.3 (first two sections), where the same (hard) instances of Table 4.2 are considered. According to these results, no dominance ex-

Table 4.2. FT-b&c without and with Fractionality Persistency (10,000 seconds time limit).

	FT-b&c					FT-b&c + FP				
	%Gap					%Gap				
Name	Root	fLB	fUB	Nodes	Time	Root	fLB	fUB	Nodes	Time
coin100.0	0.40	–	–	41	12.4	0.40	–	–	41	8.6
coin316.10	1.04	0.53	0.19	1296	10000.5	1.04	0.47	0.19	1407	10000.2
crane100.2	1.35	–	–	829	297.3	1.35	–	–	707	204.8
crane100.4	1.73	–	–	1005	210.1	1.73	–	–	751	117.6
crane316.10	0.62	0.10	0.01	2894	10000.0	0.62	0.11	0.06	2851	10000.2
rtilt100.0	0.80	–	–	417	197.2	0.80	–	–	501	227.6
rtilt100.4	0.11	–	–	13	5.4	0.11	–	–	13	4.6
rtilt316.10	0.25	0.04	0.00	1439	10000.1	0.25	–	–	1727	8887.1
stilt100.0	1.10	–	–	173	65.6	1.10	–	–	171	49.2
stilt100.3	1.73	–	–	377	157.0	1.73	–	–	325	141.3
stilt100.4	1.58	–	–	2211	1457.7	1.58	–	–	1925	1299.5
stilt316.10	2.35	1.67	19.59	1235	10000.1	2.35	1.66	19.59	1205	10000.3
balas84	1.01	–	–	93	36.3	1.01	–	–	61	15.7
balas108	1.97	–	–	215	89.8	1.97	–	–	267	89.0
balas120	1.05	–	–	662	506.2	1.05	–	–	1339	1276.3
balas160	1.26	–	–	437	522.4	1.26	–	–	737	671.1
balas200	1.24	–	–	1241	1801.2	1.24	–	–	1495	1712.8
ftv180	1.20	0.51	0.00	5226	6911.1°	1.20	–	–	939	366.0
ran1000.0	0.00	–	–	1	3.6	0.00	–	–	1	3.4
ran1000.1	0.00	–	–	3	25.6	0.00	–	–	3	23.5
ran1000.2	0.00	–	–	14	103.2	0.00	–	–	14	150.7
ran1000.3	0.01	–	–	17	98.3	0.01	–	–	11	62.2
ran1000.4	0.01	–	–	16	203.0	0.01	–	–	18	148.7
ran500.3	0.02	–	–	59	55.4	0.02	–	–	65	55.0

°execution aborted for search-tree space limit.

ists between the two transformations with respect to neither the quality of the lower bound at the root node nor the computing time. However, by considering the average behavior, the 2-node transformation seems to be preferable.

Although a fine tuning of the `Concorde` parameters is out of the scope of this section, Table 4.3 also analyzes the code sensitivity to the "chunk size" parameter (last three sections), which controls the implementation of the *local cuts* paradigm [29] used for separation (see Chapter 2 for details). In particular, setting this size to 0 ("`-C 0`") disables the generation of the "local" cuts and lets Concorde behave as a pure *STSP* code, whereas options "`-C 16`" (the default) and "`-C 24`" allow for the generation of additional cuts based on the "instance-specific" enumeration of partial solutions over vertex subsets of size up to 16 and 24, respec-

Table 4.3. Sensitivity of `Concorde` to 3- and 2-node transformations and to the chunk size parameter.

	(3-node) Concorde -s 123 -C 16					(2-node) Concorde -s 123 -C 16				
	%Gap					%Gap				
Name	Root	fLB	fUB	Nodes	Time	Root	fLB	fUB	Nodes	Time
coin100.0	0.30	–	–	25	185.7	0.29	–	–	25	106.3
coin316.10	0.80	0.31	0.06	227	10139.1	0.85	0.33	0.25	233	10101.2
crane100.2	0.69	–	–	39	347.5	0.94	–	–	41	355.5
crane100.4	1.29	–	–	23	238.9	1.27	–	–	27	277.4
crane316.10	0.34	–	–	287	5142.3	0.34	–	–	255	4287.2
rtilt100.0	0.92	–	–	23	191.7	0.94	–	–	23	129.2
rtilt100.4	0.22	–	–	15	60.8	0.27	–	–	9	26.8
rtilt316.10	0.08	–	–	27	543.3	0.12	–	–	37	593.5
stilt100.0	0.92	–	–	99	760.9	0.83	–	–	69	524.3
stilt100.3	1.49	–	–	77	769.8	1.39	–	–	75	662.5
stilt100.4	1.16	–	–	209	2271.5	1.21	–	–	179	1695.3
stilt316.10	1.99	1.40	7.68	253	10140.0	2.07	1.44	2.25	283	10132.0
balas84	1.01	–	–	31	106.8	1.01	–	–	25	78.0
balas108	2.63	–	–	497	1534.8	2.63	–	–	423	1416.0
balas120	1.05	–	–	633	5694.7	1.05	–	–	755	7186.9
balas160	1.26	–	–	541	6716.8	1.26	–	–	739	7848.0
balas200	0.50	–	–	123	1392.3	0.74	–	–	239	2294.2
ftv180	0.65	–	–	21	337.0	0.69	–	–	29	236.2
ran1000.0	0.00	–	–	61	1146.6	0.00	–	–	21	219.2
ran1000.1	0.00	–	–	35	689.7	0.00	–	–	19	191.7
ran1000.2	0.00	–	–	27	767.8	0.00	–	–	95	900.4
ran1000.3	0.01	–	–	429	7569.0	0.01	–	–	343	3977.2
ran1000.4	0.05	0.00	0.04	445	10129.7	0.01	–	–	379	3122.2
ran500.3	0.02	–	–	79	579.4	0.04	–	–	75	232.4

	(2-node) Concorde -s 123 -C 0					(2-node) Concorde -s 123 -C 24				
	%Gap					%Gap				
Name	Root	fLB	fUB	Nodes	Time	Root	fLB	fUB	Nodes	Time
coin100.0	0.82	–	–	49	88.4	0.28	–	–	15	364.5
coin316.10	1.15	0.42	0.06	567	10026.0	0.73	0.41	0.25	77	10957.5
crane100.2	2.00	–	–	141	272.0	0.44	–	–	15	1096.7
crane100.4	1.96	–	–	79	158.2	0.88	–	–	17	1148.5
crane316.10	0.98	–	–	771	5303.3	0.30	–	–	119	4931.6
rtilt100.0	1.50	–	–	59	124.6	1.04	–	–	29	836.9
rtilt100.4	0.58	–	–	9	26.4	0.17	–	–	3	257.4
rtilt316.10	0.26	–	–	51	304.5	0.06	–	–	13	2826.3
stilt100.0	1.64	–	–	203	333.3	0.70	–	–	39	2465.9
stilt100.3	2.24	–	–	535	1272.5	1.19	–	–	65	3433.6
stilt100.4	2.03	–	–	737	2350.9	0.95	–	–	129	6688.3
stilt316.10	2.27	1.74	5.44	841	10041.3	1.85	1.37	0.00	75	10663.4
balas84	1.01	–	–	17	15.2	1.01	–	–	33	547.5
balas108	2.63	–	–	1023	1269.9	2.63	–	–	343	4110.5
balas120	2.10	0.00	1.40	2849	10007.3	1.05	–	–	221	6244.1
balas160	2.02	0.25	4.03	2165	10007.1	1.01	0.00	2.27	273	10129.5
balas200	2.48	0.74	5.96	1955	10008.0	0.50	–	–	99	2918.8
ftv180	1.58	–	–	91	204.0	0.38	–	–	17	955.9
ran1000.0	0.00	–	–	11	145.2	0.00	–	–	11	186.7
ran1000.1	0.00	–	–	15	136.9	0.00	–	–	21	230.7
ran1000.2	0.00	–	–	3	79.5	0.00	–	–	33	354.1
ran1000.3	0.03	–	–	387	2836.9	0.01	–	–	393	5065.2
ran1000.4	0.01	–	–	153	957.1	0.01	–	–	177	1749.7
ran500.3	0.04	–	–	93	225.9	0.04	–	–	103	538.8

tively. In this way, we aimed at analyzing the capability of the "local" cuts method to automatically generate cuts playing an important role for the *ATSP* instance to be solved. The results of Table 4.3 show that the "local" cuts ("`-C 16`" and "`-C 24`") play a very important role, allowing one to solve to optimality difficult instances (e.g., instances `balas120`, `balas160`, and `balas200` are solvedwith chunk equal to 16, and remain unsolved without "local" cuts). Not surprisingly, the main exception concerns the uniformly-random instances (`ran1000`) for which even the *AP* bound is already very tight. Among the chunk sizes, even after the 2-node transformation, the best choiceappears to be the default ("`-C 16`") which gives the best compromise between the separation overhead and the lower bound improvement. We have also tested chunk sizes 8 and 32, without obtaining better results. (The 3-node transformation exhibits a similar behavior.)

5.2. Code Comparison

A comparison of the four algorithms, namely, `CDT`, `FT-add` (modified version), `FT-b&c` ("`+ FP`" version) and `Concorde` ("`(2-node) -C 16`" version), is given in Tables 4.4, 4.5, 4.6 and 4.7 on the complete set of 86 instances. As to time limit, we imposed 1,000 CPU seconds for `CDT` and `FT-add`, and 10,000 CPU seconds for `FT-b&c` and `Concorde`. The smaller time limit given to the first two algorithms is chosen so as to keep the required search-tree space reasonable, but it does not affect the comparison with the other algorithms: according to preliminary computational experiments on some hard instances, either the branch-and-bound approaches solve an instance within 1,000 CPU seconds, or the final gap is too large to hope in a convergence within 10,000 seconds.

As to the lower bound at the root node, the tables show that the additive approach obtains significantly better results than the *AP* lower bound, but is dominated by both cutting plane approaches. In its pure *STSP* version ("`-C 0`"), the `Concorde` codeobtains a root-node lower bound which is dominated by the `FT-b&c` one, thus showing the effectiveness of addressing the *ATSP* in its original (directed) version. Of course, one can expect to improve the performance of `FT-b&c` by exploiting additional classes of *ATSP*-specific cuts such as the lifted cycle inequalities described in Chapter 3. As to `Concorde`, we observe that the use of the "local" cuts leads to a considerable improvement of the root-node lower bound. Not surprisingly, this improvement appears more substantial than in the case of pure *STSP* instances: in our view, this is again an indication of the importance of exploiting the structure of the original asymmetric problem, which results into a very special structure

Table 4.4. Comparison of branch-and-bound codes. Time limit of 1,000 seconds.

	CDT					FT-add				
	%Gap					%Gap				
Name	Root	fLB	fUB	Nodes	Time	Root	fLB	fUB	Nodes	Time
coin100.0	12.93	5.41	23.17	192324	1000.0	4.28	0.77	0.19	185361	1000.0
coin100.1	15.66	9.62	21.51	210379	1000.0	3.68	1.89	1.51	154925	1000.0
coin100.2	14.57	7.65	18.40	215216	1000.0	5.24	1.82	0.36	162688	1000.0
coin100.3	20.68	12.52	20.49	214915	1000.0	5.02	1.33	0.57	166505	1000.0
coin100.4	13.03	6.70	14.37	219186	1000.0	5.91	1.34	1.72	167641	1000.0
coin316.10	14.94	12.22	19.14	229130	1000.0	6.96	5.19	14.01	13470	1000.0
crane100.0	8.55	1.58	8.08	164790	1000.0	4.09	0.19	0.00	168273	1000.0
crane100.1	5.57	1.48	3.43	170445	1000.0	3.12	0.16	0.00	170634	1000.0
crane100.2	10.30	2.41	8.41	165701	1000.0	5.84	1.06	0.37	103956	1000.0
crane100.3	7.02	0.65	3.87	155420	1000.0	3.38	–	–	8643	40.7
crane100.4	8.69	4.20	13.79	162903	1000.0	4.78	2.11	2.15	96990	1000.0
crane316.10	8.75	5.76	8.14	180724	1000.0	3.66	2.93	1.93	9940	1000.3
disk100.0	3.00	–	–	5360	0.7	2.31	–	–	538	0.9
disk100.1	4.34	0.41	1.81	151974	1000.0	2.81	–	–	12706	29.6
disk100.2	2.75	–	–	465	0.0	2.17	–	–	167	0.7
disk100.3	1.33	–	–	4909	0.7	1.28	–	–	738	1.5
disk100.4	2.12	–	–	143	0.0	1.47	–	–	74	0.3
disk316.10	1.21	–	–	3397	2.1	0.76	–	–	502	43.5
rtilt100.0	22.59	13.19	10.62	238487	1000.0	7.52	2.93	3.00	96426	1000.0
rtilt100.1	20.83	12.29	18.83	224840	1000.0	7.38	2.73	1.98	109535	1000.0
rtilt100.2	23.34	13.36	18.82	215193	1000.0	6.73	2.64	3.30	105315	1000.0
rtilt100.3	18.86	10.28	26.64	247630	1000.0	3.27	1.76	1.13	115796	1000.0
rtilt100.4	13.52	7.11	14.05	213573	1000.0	3.89	0.83	0.94	108540	1000.0
rtilt316.10	18.39	15.14	19.11	207103	1000.0	9.07	6.06	11.95	8314	1000.2
shop100.0	0.28	–	–	64	0.0	0.14	–	–	81	0.4
shop100.1	0.67	–	–	3234	1.0	0.38	–	–	587	2.8
shop100.2	0.98	–	–	1345	0.4	0.41	–	–	990	5.2
shop100.3	0.36	–	–	604	0.2	0.17	–	–	872	2.9
shop100.4	0.35	–	–	217	0.1	0.28	–	–	273	1.3
shop316.10	0.16	–	–	1228	3.1	0.11	–	–	661	91.1
stilt100.0	22.15	11.36	21.47	213441	1000.0	9.32	2.52	1.62	97255	1000.0
stilt100.1	19.34	10.55	24.20	213919	1000.0	6.49	2.78	2.49	103875	1000.0
stilt100.2	23.61	11.21	24.43	199478	1000.0	9.81	3.23	2.84	110778	1000.0
stilt100.3	17.30	8.86	24.97	198401	1000.0	7.05	3.77	2.91	116219	1000.0
stilt100.4	13.19	7.35	22.05	202776	1000.0	6.93	2.19	0.60	99972	1000.0
stilt316.10	16.36	14.05	24.21	174030	1000.0	9.82	6.70	16.39	6508	1000.1
super100.0	0.25	–	–	36	0.0	0.13	–	–	11	0.1
super100.1	1.28	–	–	38	0.0	0.26	–	–	72	0.2
super100.2	1.79	–	–	169	0.1	0.64	–	–	68	0.2
super100.3	0.90	–	–	69	0.1	0.39	–	–	122	0.3
super100.4	0.25	–	–	6	0.0	0.25	–	–	16	0.1
super316.10	1.33	0.57	3.37	103934	1000.0	0.52	–	–	8515	236.7

Table 4.5. Comparison of branch-and-cut codes. Time limit of 10,000 seconds.

	FT-b&c + FP					(2-node) Concorde -s 123 -C 16				
	%Gap					%Gap				
Name	Root	fLB	fUB	Nodes	Time	Root	fLB	fUB	Nodes	Time
coin100.0	0.40	–	–	41	8.6	0.29	–	–	25	106.3
coin100.1	0.47	–	–	29	5.9	0.30	–	–	7	49.1
coin100.2	0.45	–	–	25	9.1	0.13	–	–	3	40.4
coin100.3	0.19	–	–	9	2.8	0.08	–	–	3	21.2
coin100.4	0.48	–	–	17	4.9	1.05	–	–	7	64.1
coin316.10	1.04	0.47	0.19	1407	10000.2	0.85	0.33	0.25	233	10101.2
crane100.0	0.32	–	–	7	1.4	0.00	–	–	1	6.5
crane100.1	0.28	–	–	7	1.5	0.01	–	–	3	15.9
crane100.2	1.35	–	–	707	204.8	0.94	–	–	41	355.5
crane100.3	0.03	–	–	3	0.5	0.00	–	–	1	3.4
crane100.4	1.73	–	–	751	117.6	1.27	–	–	27	277.4
crane316.10	0.62	0.11	0.06	2851	10000.2	0.34	–	–	255	4287.2
disk100.0	0.22	–	–	11	1.0	0.14	–	–	3	7.1
disk100.1	0.35	–	–	29	2.9	0.32	–	–	21	40.4
disk100.2	0.04	–	–	3	0.3	0.04	–	–	3	5.6
disk100.3	0.00	–	–	1	0.1	0.00	–	–	1	2.5
disk100.4	0.00	–	–	3	0.3	0.15	–	–	3	9.3
disk316.10	0.03	–	–	5	5.6	0.04	–	–	7	31.1
rtilt100.0	0.80	–	–	501	227.6	0.94	–	–	23	129.2
rtilt100.1	0.24	–	–	23	6.8	0.09	–	–	5	26.7
rtilt100.2	0.00	–	–	1	0.4	0.00	–	–	1	8.1
rtilt100.3	0.32	–	–	19	4.2	0.00	–	–	1	19.3
rtilt100.4	0.11	–	–	13	4.6	0.27	–	–	9	26.8
rtilt316.10	0.25	–	–	1727	8887.1	0.12	–	–	37	593.5
shop100.0	0.01	–	–	7	0.7	0.03	–	–	13	28.8
shop100.1	0.02	–	–	13	1.4	0.02	–	–	17	39.7
shop100.2	0.02	–	–	11	1.1	0.02	–	–	7	17.7
shop100.3	0.03	–	–	17	1.6	0.03	–	–	15	24.4
shop100.4	0.02	–	–	5	0.5	0.01	–	–	9	16.7
shop316.10	0.01	–	–	17	14.3	0.01	–	–	15	122.2
stilt100.0	1.10	–	–	171	49.2	0.83	–	–	69	524.3
stilt100.1	1.29	–	–	167	57.1	1.13	–	–	23	221.4
stilt100.2	0.57	–	–	15	6.1	0.22	–	–	5	56.9
stilt100.3	1.73	–	–	325	141.3	1.39	–	–	75	662.5
stilt100.4	1.58	–	–	1925	1299.5	1.21	–	–	179	1695.3
stilt316.10	2.35	1.66	19.59	1205	10000.3	2.77	1.44	2.25	283	10132.0
super100.0	0.00	–	–	6	0.5	0.00	–	–	7	6.0
super100.1	0.00	–	–	1	0.2	0.00	–	–	9	8.5
super100.2	0.00	–	–	1	0.1	0.00	–	–	1	1.7
super100.3	0.00	–	–	1	0.2	0.00	–	–	1	3.5
super100.4	0.00	–	–	1	0.1	0.00	–	–	5	6.1
super316.10	0.00	–	–	4	3.9	0.00	–	–	1	14.9

Table 4.6. Comparison of branch-and-bound codes. Time limit of 1,000 seconds.

	CDT					FT-add				
	%Gap					%Gap				
Name	Root	fLB	fUB	Nodes	Time	Root	fLB	fUB	Nodes	Time
balas84	14.07	5.53	34.17	192409	1000.0	5.53	–	–	612994	986.6
balas108	25.00	14.47	53.95	166420	1000.0	9.87	1.97	0.66	164537	1000.0
balas120	21.68	12.24	36.71	154920	1000.0	13.29	7.34	7.34	122080	1000.0
balas160	19.40	10.58	43.83	148006	1000.0	11.34	8.31	13.35	70260	1000.0
balas200	15.63	12.66	47.64	141847	1000.0	8.68	6.70	21.09	47711	1000.0
ftv180	5.07	0.65	2.06	150773	1000.0	4.28	0.62	0.10	64938	1000.0
uk66	72.91	5.84	35.51	193383	1000.0	31.71	–	–	50005	152.1
ran1000.0	0.00	–	–	61	0.7	0.00	–	–	811	94.2
ran1000.1	0.05	–	–	50	0.7	0.05	–	–	97	9.1
ran1000.2	0.09	–	–	182	3.9	0.09	–	–	1022	90.0
ran1000.3	0.05	–	–	73	1.1	0.05	–	–	795	42.9
ran1000.4	0.07	–	–	87	1.5	0.07	–	–	577	48.5
ran500.0	0.00	–	–	15	0.1	0.00	–	–	1	0.3
ran500.1	0.06	–	–	11	0.1	0.06	–	–	12	1.1
ran500.2	0.15	–	–	58	0.2	0.15	–	–	160	3.1
ran500.3	0.06	–	–	52	0.1	0.06	–	–	263	6.6
ran500.4	0.07	–	–	37	0.1	0.07	–	–	85	4.6
br17	100.00	–	–	43879	3.6	0.00	–	–	11	0.0
ft53	14.11	5.53	15.28	154920	1000.0	1.56	–	–	131	0.2
ft70	1.80	–	–	2624	0.4	0.57	–	–	220	0.3
ftv33	7.85	–	–	81	0.0	3.73	–	–	120	0.1
ftv35	6.25	–	–	428	0.0	3.53	–	–	647	0.2
ftv38	6.01	–	–	424	0.0	3.01	–	–	854	0.3
ftv44	5.70	–	–	214	0.0	4.46	–	–	289	0.1
ftv47	6.98	–	–	585	0.1	3.49	–	–	1026	0.4
ftv55	10.76	–	–	6989	1.1	6.59	–	–	3253	1.5
ftv64	6.42	–	–	4552	0.8	4.89	–	–	2553	1.3
ftv70	9.44	–	–	10328	3.3	6.92	–	–	4347	3.7
ftv90	6.33	–	–	4220	0.7	3.17	–	–	1670	2.5
ftv100	6.60	–	–	27700	16.6	3.91	–	–	11089	29.6
ftv110	5.87	–	–	14393	5.0	4.49	–	–	12931	38.4
ftv120	6.51	–	–	35382	50.1	5.77	–	–	32600	99.4
ftv130	4.46	–	–	5888	6.4	3.77	–	–	3512	16.6
ftv140	4.92	–	–	11128	13.9	4.26	–	–	7083	38.8
ftv150	3.91	–	–	2645	3.1	3.03	–	–	2778	17.0
ftv160	4.58	–	–	49169	93.8	4.03	–	–	38007	316.2
ftv170	4.50	0.36	1.96	155326	1000.0	4.14	0.18	0.00	85236	1000.0
kro124p	6.22	1.20	10.95	166664	1000.0	2.73	–	–	18909	135.7
p43	97.37	95.62	0.09	186698	1000.0	0.37	0.30	0.02	321417	1000.0
rbg323	0.00	–	–	1	0.1	0.00	–	–	1	0.3
rbg358	0.00	–	–	1	0.1	0.00	–	–	1	0.5
rbg403	0.00	–	–	1	0.1	0.00	–	–	1	1.0
rbg443	0.00	–	–	1	0.1	0.00	–	–	1	1.2
ry48p	13.21	1.27	0.00	196577	1000.0	2.94	–	–	22825	20.3

Table 4.7. Comparison of branch-and-cut codes. Time limit of 10,000 seconds.

	FT-b&c + FP					(2-node) Concorde -s 123 -C 16				
	%Gap					%Gap				
Name	Root	fLB	fUB	Nodes	Time	Root	fLB	fUB	Nodes	Time
balas84	1.01	–	–	61	15.7	1.01	–	–	25	78.0
balas108	1.97	–	–	267	89.0	2.63	–	–	423	1416.0
balas120	1.05	–	–	1339	1276.3	1.05	–	–	755	7186.9
balas160	1.26	–	–	737	671.1	1.26	–	–	739	7848.0
balas200	1.24	–	–	1495	1712.8	0.74	–	–	239	2294.2
ftv180	1.20	–	–	939	366.0	0.69	–	–	29	236.2
uk66	2.29	–	–	47	5.7	1.47	–	–	17	66.1
ran1000.0	0.00	–	–	1	3.4	0.00	–	–	21	219.2
ran1000.1	0.00	–	–	3	23.5	0.00	–	–	19	191.7
ran1000.2	0.00	–	–	14	150.7	0.00	–	–	95	900.4
ran1000.3	0.01	–	–	11	62.2	0.01	–	–	343	3977.2
ran1000.4	0.01	–	–	18	148.7	0.01	–	–	379	3122.2
ran500.0	0.00	–	–	1	0.8	0.00	–	–	5	29.2
ran500.1	0.00	–	–	1	1.8	0.00	–	–	1	20.0
ran500.2	0.00	–	–	4	31.4	0.00	–	–	11	52.7
ran500.3	0.02	–	–	65	55.0	0.04	–	–	75	232.4
ran500.4	0.00	–	–	2	3.4	0.02	–	–	15	53.8
br17	0.00	–	–	1	0.0	0.00	–	–	1	0.2
ft53	0.00	–	–	1	0.1	0.00	–	–	1	0.6
ft70	0.02	–	–	5	0.2	0.01	–	–	3	3.2
ftv33	0.00	–	–	1	0.0	0.00	–	–	1	0.3
ftv35	0.88	–	–	9	0.4	0.68	–	–	5	9.0
ftv38	0.85	–	–	13	0.6	0.52	–	–	5	14.5
ftv44	0.37	–	–	9	0.5	0.12	–	–	3	9.1
ftv47	1.01	–	–	11	0.5	0.62	–	–	11	23.4
ftv55	0.81	–	–	35	1.4	0.44	–	–	3	9.0
ftv64	1.36	–	–	29	2.6	0.33	–	–	7	20.8
ftv70	0.92	–	–	11	1.1	0.26	–	–	3	17.8
ftv90	0.25	–	–	5	0.5	0.06	–	–	3	14.7
ftv100	0.39	–	–	21	2.2	0.00	–	–	1	12.6
ftv110	0.77	–	–	77	7.4	0.05	–	–	3	25.6
ftv120	0.97	–	–	123	13.1	0.28	–	–	7	54.4
ftv130	0.35	–	–	7	1.6	0.00	–	–	1	16.6
ftv140	0.25	–	–	9	2.1	0.00	–	–	3	25.6
ftv150	0.27	–	–	21	2.6	0.00	–	–	5	27.0
ftv160	0.67	–	–	17	3.8	0.30	–	–	7	55.7
ftv170	0.87	–	–	15	4.1	0.40	–	–	3	41.9
kro124p	0.04	–	–	3	1.0	0.00	–	–	1	9.9
p43	0.16	–	–	141	9.3	0.16	–	–	13	22.7
rbg323	0.00	–	–	1	0.4	0.00	–	–	3	23.9
rbg358	0.00	–	–	1	0.5	0.00	–	–	3	29.3
rbg403	0.00	–	–	1	1.3	0.00	–	–	5	49.3
rbg443	0.00	–	–	1	1.4	0.00	–	–	3	34.5
ry48p	0.53	–	–	7	0.8	0.35	–	–	5	22.9

of its *STSP* counterpart which is not captured adequately by the usual classes of *STSP* cuts (comb, clique tree inequalities, etc.).

As to the overall computing time, for the randomly generated instances (`ran`) the mosteffective code is `CDT`, which also performs very well on `shop` and `rbg` instances. Code `FT-add` has, on average,better performance than `CDT` (solving to optimality, within 1,000 CPU seconds, 8 more instances), never being, however, the best of the four codes on any instance.

Codes `FT-b&c` and `Concorde` turn out to be, in general, the most effective approaches, the first code being almost always faster than the second one, though it often requires more branching nodes. We believe this is mainly due to the faster (*ATSP*-specific) separation and pricing tools used in `FT-b&c`. Strangely enough, however, `FT-b&c` does not solve to optimality instance `crane316.10` which is, instead, solved by `Concorde` even in its pure *STSP* version "–C 0": see Table 4.3. This pathological behavior of FT–b&c seems to derive from an unlucky sequence of wrong choices of the branching variable in the first levels of the branching tree.

Not surprisingly, the `Concorde` implementation proved very robust for hard instances of large size, as it has been designed and engineered to address very large *STSP* instances. On the other hand, `FT-b&c` was implemented by its authors to solve medium-size *ATSP* instances with up to 200 vertices, and exploits neither sophisticated primal heuristics nor optimized interfaces with the LP solver. In our view, the fact that its performance is comparable or better than that of `Concorde "-C 16"` (and considerably better than that of the pure *STSP* `Concorde "-C 0"`) is mainly due to the effectiveness of the *ATSP*-specific separation procedures used. This suggests that enriching the Concorde arsenal of *STSP* separation tools by means of *ATSP*-specific separation procedures would be the road to go for the solution of hard *ATSP* instances.

Acknowledgments

Work supported by Ministero dell'Istruzione, dell'Università e della Ricerca (M.I.U.R) and by Consiglio Nazionale delle Ricerche (C.N.R.), Italy. The computational experiments have been executed at the Laboratory of Operations Research of the University of Bologna (Lab.O.R.).

References

[1] J. Campbell, "Speaker recognition: a tutorial," *Proc. IEEE*, vol. 85, pp. 1437–1462, Sept. 1997.

[2] D. A. Reynolds, T. Quatieri, and R. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1–3, pp. 19–41, 2000.

[3] D. A. Reynolds, "Comparison of background normalization methods for text-independent speaker verification," in *Proc. Eurospeech*, 1997.

[4] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Trans. Speech Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[5] J. L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains," *IEEE Trans. Speech Audio Processing*, vol. 2, pp. 291–298, Apr. 1994.

[6] E. Bocchieri, "Vector quantization for the efficient computation of continuous density likelihoods," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1993, pp. 692–695.

[7] K. M. Knill, M. J. F. Gales, and S. J. Young, "Use of Gaussian selection in large vocabulary continuous speech recognition using HMMs," in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[8] D. B. Paul, "An investigation of Gaussian shortlists," in *Proc. Automatic Speech Recognition and Understanding Workshop*, 1999.

[9] T. Watanabe, K. Shinoda, K. Takagi, and K.-I. Iso, "High speed speech recognition using tree-structured probability density function," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1995.

[10] J. Simonin, L. Delphin-Poulat, and G. Damnati, "Gaussian density tree structure in a multi-Gaussian HMM-based speech recognition system,"

[11] T. J. Hanzen and A. K. Halberstadt, "Using aggregation to improve the performance of mixture Gaussian acoustic models," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1998.

[12] M. Padmanabhan, L. R. ahl, and D. Nahamoo, "Partitioning the feature space of a classifier with linear hyperplanes," *IEEE Trans. Speech Audio Processing*, vol. 7, no. 3, pp. 282–288, 1999. in *Proc. Int. Conf. Spoken Language Processing*, 1998.

[13] R. Auckenthaler and J. Mason, "Gaussian selection applied to text-independent speaker verification," in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[14] J. McLaughlin, D. Reynolds, and T. Gleason, "A study of computation speed-ups of the GMM-UBM speaker recognition system," in *Proc. Eurospeech*, 1999.

[15] S. van Vuuren and H. Hermansky, "On the importance of components of the modulation spectrum of speaker verification," in *Proc. Int. Conf. Spoken Language Processing*, 1998.

[16] B. L. Pellom and J. H. L. Hansen, "An efficient scoring algorithm for Gaussian mixture model based speaker identification," *IEEE Signal Processing Lett.*, vol. 5, no. 11, pp. 281–284, 1998.

[17] J. Oglesby and J. S. Mason, "Optimization of neural models for speaker identification," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1990, pp. 261–264.

[18] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global optimization of a neural network—hidden Markov model hybrid," *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 252–259, 1992.

[19] H. Bourlard and C. J. Wellekins, "Links between Markov models and multilayer perceptrons," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 1167–1178, Dec. 1990.

[20] J. Navrátil, U. V. Chaudhari, and G. N. Ramaswamy, "Speaker verification using target and background dependent linear transforms and multi-system fusion," in *Proc. Eurospeech*, 2001.

[21] L. P. Heck, Y. Konig, M. K. Sonmez, and M. Weintraub, "Robustness to telephone handset distortion in speaker recognition by discriminative feature design," *Speech Commun.*, vol. 31, pp. 181–192, 2000.

[22] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Statist. Soc.*, vol. 39, pp. 1–38, 1977.

[23] K. Shinoda and C. H. Lee, "A structural Bayes approach to speaker adaptation," *IEEE Trans. Speech Audio Processing*, vol. 9, no. 3, pp. 276–287, 2001.

[24] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1990.

[25] J. C. Junqua, *Robust Speech Recogntion in Embedded Systems and PC*

[26] U. V. Chaudhari, J. Navrátil, S. H. Maes, and R. A. Gopinath, "Transformation enhanced multi-grained modeling for text-independent speaker recognition," in *Proc. Int. Conf. Spoken Language Processing*, 2000.

[27] Q. Lin, E.-E. Jan, C. W. Che, D.-S. Yuk, and J. Flanagan, "Selective use of the speech spectrum and a VQGMM method for speaker identification," in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[28] S. Raudys, *Statistical and Neural Classifiers: An Integrated Approach to Design*. New York: Springer, 2001.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, pp. 318–364.

[30] [Online] Available: http://www.nist.gov/speech/tests/spk/index.htm.

[31] J. Pelecanos and S. Sridharan, "Feature warping for robust speaker verification," in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[32] B. Xiang, U. V. Chaudhari, J. Navrátil, N. Ramaswamy, and R. A. Gopinath, "Short-time Gaussianization for robust speaker verification," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 2002.

[33] G. R. Doddington, M. A. Przybocki, A. F. Martin, and D. A. Reynolds, "The NIST speaker recognition evaluation—overview, methodology, systems, results, perspective," *Speech Communication*, vol. 31, pp. 225–254, 2000.

Bing Xiang (M'03) was born in 1973 in China. He received the B.S. degree in radio and electronics and M.E. degree in signal and information processing from Peking University in 1995 and 1998, respectively. In January, 2003, he received the Ph.D. degree in electrical engineering from Cornell University, Ithaca, NY.

From 1995 to 1998, he worked on speaker recognition and auditory modeling in National Laboratory on Machine Perception, Peking University. Then he entered Cornell University and worked on speaker recognition and speech recognition in DISCOVER Lab as a Research Assistant. He also worked in the Human Language Technology Department of IBM Thomas J. Watson Research Center as a summer intern in both 2000 and 2001. He was a selected remote member of the SuperSID Group in the 2002 Johns Hopkins CLSP summer workshop in which he worked on speaker verification with high-lelvel information. In January, 2003, he joined the Speech and Language Processing Department of BBN Technologies where he is presently a Senior Staff Consultant-Technology. His research interests include large vocabulary speech recognition, speaker recognition, speech synthesis, keyword spotting, neural networks and statistical pattern recognition.

Toby Berger (S'60–M'66–SM'74–F'78) was born in New York, NY, on September 4, 1940. He received the B.E. degree in electrical engineering from Yale University, New Haven, CT in 1962, and the M.S. and Ph.D. degrees in applied mathematics from Harvard University, Cambridge, MA in 1964 and 1966, respectively.

From 1962 to 1968 he was a Senior Scientist at Raytheon Company, Wayland, MA, specializing in communication theory, information theory, and coherent signal processing. In 1968 he joined the faculty of Cornell University, Ithaca, NY where he is presently the Irwin and Joan Jacobs Professor of Engineering. His research interests include information theory, random fields, communication networks, wireless communications, video compression, voice and signature compression and verification, neuroinformation theory, quantum information theory, and coherent signal processing. He is the author/co-author of Rate Distortion Theory: A Mathematical Basis for Data Compression, Digital Compression for Multimedia: Principles and Standards, and Information Measures for Discrete Random Fields.

Dr. Berger has served as editor-in-chief of the IEEE TRANSACTIONS ON INFORMATION THEORY and as president of the IEEE Information Theory