# Advanced analysis of branch and bound algorithms

Turkensteen, Marcel

# Chapter 3

# Tolerance-based Branch and Bound Algorithms for the ATSP

## 3.1 Introduction

The Traveling Salesman Problem (TSP) is the problem of finding a shortest tour through a given number of locations such that every location is visited exactly once. The cost of traveling from location $i$ to location $j$ is denoted by $c(i, j)$. These costs are called *symmetric* if $c(i, j) = c(j, i)$ for each pair of cities $i$ and $j$, and *asymmetric* otherwise. The fact that the TSP is a typical $\mathcal{NP}$-hard optimization problem means, roughly spoken, that solving instances with a large number of cities is very difficult if not impossible. Recent developments in polyhedral theory and heuristics have significantly increased the size of instances which can be solved to optimality. The best known exact algorithms are based on either the Branch and Bound (BnB) method for the Asymmetric TSP (ATSP) (Fischetti *et al.*, 2002) or the Branch-and-Cut method for the Symmetric TSP (STSP) using the double index formulation of the problem (see Naddef (2002)).

Currently, most algorithms for the TSP delete high cost arcs or edges and save the low cost ones. A drawback of this strategy is that costs of arcs and

[0]This chapter is based on the article: M. Turkensteen, D. Ghosh, B. Goldengorin, and G. Sierksma,"Tolerance-Based Branch and Bound Algorithms for the ATSP", accepted for publication in EJOR. A preliminary version of this paper has been published in B. Goldengorin, G. Sierksma, M. Turkensteen, "Tolerance Based Algorithms for the ATSP", Graph-Theoretic Concepts in Computer Science. 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004. Hromkovic J., Nagl M., Westfechtel B. (eds.), Lecture Notes in Computer Science 3353, pp. 222–234, (2004).

edges are no accurate indicators whether those arcs or edges are saved in an optimal TSP solution. In this paper, it is shown that *tolerances* are better indicators. A tolerance value of an edge/arc is the cost of excluding or including that edge/arc from the solution at hand; see Section 3.3. Although the concept of tolerances has been applied for decades (in sensitivity analysis; see for example Libura *et al.* (1998); Lin and Wen (2003)), only Helsgaun's version of the Lin-Kernighan heuristic for the STSP applies tolerances; see Helsgaun (2000).

We apply upper tolerances in BnB algorithms for the ATSP. A BnB algorithm initially solves a *relaxation* of the original hard problem. In case of the ATSP, the Assignment Problem (AP) is a common choice. The AP is the problem of assigning $n$ people to $n$ jobs against minimum cost; an optimal solution of the AP is called a *minimum cycle cover*. If the minimum cycle cover at hand is a complete tour, then the ATSP instance is solved; otherwise, the problem is partitioned into new subproblems by including and excluding arcs. In the course of the process, a *search tree* is generated in which all solved subproblems are listed. BnB algorithms comprise two major steps: *branching* and *bounding*.

The objective of *branching* is to find a good, or even optimal, ATSP solution in an effective way. If the current AP solution is infeasible, then there may exist a subset of elements of this solution, the so-called *survival set*, which also appears in an optimal TSP solution eventually obtained by the BnB algorithm. An effective BnB algorithm cherishes arcs in survival sets and disposes of the other ones, the *extinction arcs*. Obviously, survival sets are not known beforehand. Predictions of what arcs belong to the survival set are usually based on the arc costs. We claim that the predictions are much more accurate if upper tolerance values of arcs are used instead; see Section 3.5.

The objective of *bounding* is to fathom as many nodes in the search tree as possible. A subproblem is fathomed if its lower bound exceeds the value of the best solution found so far in the process. An AP solution is infeasible for the ATSP, if it consists of two or more subcycles; we call such subcycles *offenders*. To obtain a complete tour, at least one offender must be "broken", meaning that its arcs are successively prohibited in the next stage of the process. Since the cost of removing an arc is its upper tolerance value, upper tolerance values provide us with the cost of breaking an offender, and hence, they can be used to tighten the lower bounds. The higher the lower bound, the larger the set of subproblems

that are fathomed; see Section 3.6.

Compared to their cost-based counterparts, tolerance-based BnB algorithms have one big drawback: whereas cost values need to be looked up, tolerance values must be calculated. So the question is whether the reduction in the size of the search tree is on average sufficiently large to compensate for the additional tolerance computation times. Computational experiments, performed in Section 3.9, show that it is so for random, sparse, and various ATSPLIB instances. The conclusions and future research directions appear in Section 3.10.

## 3.2 Branch and Bound Algorithms for the Asymmetric Traveling Salesman Problem

ATSP instances are often solved to optimality with BnB algorithms that take the Assignment Problem (AP) as a relaxation (Fischetti *et al.*, 2002). Instead of a single tour through all cities, an optimal solution of the AP usually consists of more than one tour, the so-called *subcycles*. The AP-based BnB algorithms then remove the arcs in a chosen subcycle one by one, combining the subcycles into an optimal tour $H^*$ of the given instance. BnB algorithms are built up from the following four basic ingredients; see for example Miller and Pekny (1991).

The *branching rule* prescribes how the current problem should be partitioned into subproblems. An effective branching rule for the ATSP with the AP relaxation is introduced in Carpaneto and Toth (1980).

The *search strategy* prescribes which subproblems should be expanded next. Two widely used strategies are *Depth First Search (DFS)* which solves the most recently generated subproblem first, and *Best First Search (BFS)* which solves the most promising subproblem first, i.e., the subproblem with the lowest value of the AP bound.

The *upper bounding strategy* prescribes how tours should be constructed in the BnB process. A method to construct feasible ATSP tours from AP solutions is the patching procedure by Karp and Steele (1990).

The *lower bounding strategy* determines how a lower bound of the solution value of any subproblem should be constructed. The value of the AP solu-

tion of the subproblem is usually taken as a lower bound.

State-of-the-art BnB algorithms for the ATSP can be found in Miller and Pekny (1991) and Carpaneto *et al.* (1995). These algorithms apply patching to obtain upper bounds, use AP lower bounds, and branch on a smallest cycle in the current AP solution, i.e., a cycle of smallest cardinality. The search strategy of both algorithms is BFS. This means that for many ATSP instances, solutions are obtained in very short solution times. On the other hand, a list of subproblems should be maintained in order to determine the most promising one. As a consequence, BFS BnB algorithms tend to run out of memory when the search trees grow large. For example, Miller and Pekny (1991) report that their BnB algorithm cannot solve symmetric instances of size as small as 30! The DFS strategy has two advantages over BFS algorithms: less memory overhead is required to the store data on the search tree, and the relaxation solution of the parent node in the search can be used to obtain a relaxation solution at the current node more quickly. As a consequence, it can be expected that DFS algorithms are better applicable to difficult instances. For this reason, we consider DFS algorithms in this paper, and compare the performance of our DFS algorithms to the algorithm by Carpaneto *et al.* (1995).

## 3.3 Tolerances for Combinatorial Optimization Problems

In this section, we introduce the notion of upper and lower tolerances in case of a general Combinatorial Optimization Problem (COP). A *Combinatorial Optimization Problem* COP$(\mathcal{E}, C, \mathcal{D}, f_C)$ is the problem of finding a solution

$$S^* \in \arg\, opt\{f_C(S) \mid S \in \mathcal{D}\},$$

where $C : \mathcal{E} \rightarrow \Re$ is the given *instance* of the problem with *ground set* $\mathcal{E}$ satisfying $|\mathcal{E}| = m$ $(m \geq 1)$, $\mathcal{D} \subseteq 2^{\mathcal{E}}$ is the *set of feasible solutions*, and $f_C : 2^{\mathcal{E}} \rightarrow \Re$ is the *objective function* of the problem. $\mathcal{D}^* = \arg\, opt\{f_C(S) \mid S \in \mathcal{D}\}$ is the set of optimal solutions. It is assumed that $\mathcal{D}^* \neq \emptyset$. In the remaining part of this paper we take $opt = \min$.

Let $g \in \mathcal{E}$, and $\alpha \geq 0$. By $C_{\alpha,g} : \mathcal{E} \rightarrow \Re$ we denote the instance defined as $C_{\alpha,g}(e) = C(e)$ for each $e \in \mathcal{E} \setminus \{g\}$, and $C_{\alpha,g}(g) = C(g) + \alpha$. Take any

$S^* \in \mathcal{D}^*$, and $e \in \mathcal{E}$. The *upper tolerance* of $e$ with respect to $S^*$ is denoted and defined as

$$u_{S^*}(e) = \max\{\alpha \geq 0 \,:\, S^* \in \arg\min\{f_{C_{\alpha,e}}(S) \,:\, S \in \mathcal{D}\}\},$$

and the *lower tolerance* of $e$ with respect to $S^*$ as

$$l_{S^*}(e) = \max\{\alpha \geq 0 \,:\, S^* \in \arg\min\{f_{C_{-\alpha,e}}(S) \,:\, S \in \mathcal{D}\}\}.$$

I.e. $u_{S^*}(e)$ is the maximal increase of $C(e)$ under which $S^*$ stays optimal, and $l_{S^*}(e)$ the maximal decrease of $C(e)$ under which $S^*$ stays optimal.

We assume that $f_C$ is *monotone*, meaning that for each $S \in 2^{\mathcal{E}}$ and each $\alpha \geq 0$, it holds that

$$f_{C_{\alpha,e}}(S) \geq f_{C_{0,e}}(S).$$

*Sum functions* with $f_C(S) = \sum_{e \in S} C(e)$, *bottleneck functions* with $f_C(S) = \max_{e \in S} C(e)$, and *product functions* with $f_C(S) = \prod_{e \in S} C(e)$ and $C(e) \geq 1$ for each $e \in \mathcal{E}$ are all monotone functions.

We call the set $\mathcal{D}$ of feasible solutions *non-embedded* if for each $S_1, S_2 \in \mathcal{D}$ with $S_1 \neq S_2$, it holds that neither $S_1 \subset S_2$ nor $S_2 \subset S_1$.

The following theorem, of which the proof is left to the reader, can be seen as a straightforward generalization of Libura's theorem on tolerances (see Libura (1991)) for the TSP. We will use the following extra notations. Let $e \in \mathcal{E}$. Then $\mathcal{D}_+(e) = \{S \in \mathcal{D} \,:\, e \in S\}$, and $\mathcal{D}_-(e) = \{S \in \mathcal{D} \,:\, e \notin S\}$. Clearly, $\mathcal{D} = \mathcal{D}_-(e) \cup \mathcal{D}_+(e)$ and $\mathcal{D}_-(e) \cap \mathcal{D}_+(e) = \emptyset$. $\mathcal{D}^*_+(e)$ and $\mathcal{D}^*_-(e)$ are the sets of optimal solutions containing $e$ and not containing $e$, respectively.

**Theorem 3.3.1.** *Consider COP $(\mathcal{E}, C, \mathcal{D}, f_C)$ with monotone $f_C$. For each $S^* \in \mathcal{D}^*$, the following holds:*

1. $e \in \cap\mathcal{D}^*$ *iff*
   $u_{S^*}(e) = f_C(S) - f_C(S^*) > 0$
   *for each* $S \in \mathcal{D}^*_-(e)$,
   $l_{S^*}(e) = \infty$;
2. $e \in \mathcal{E} \setminus \cup\mathcal{D}^*$ *iff*
   $u_{S^*}(e) = \infty$, $l_{S^*}(e) = f_C(S) - f_C(S^*) > 0$
   *for each* $S \in \mathcal{D}^*_+(e)$;
3. $e \in S^* \setminus \cap\mathcal{D}^*$ *iff*
   $u_{S^*}(e) = 0$, $l_{S^*}(e) = \infty$;
4. $e \in \cup\mathcal{D}^* \setminus S^*$ *iff*
   $u_{S^*}(e) = \infty$, $l_{S^*}(e) = 0$.

If $|\mathcal{D}^*| = 1$, then this theorem boils down to Libura's theorem on tolerances. If $\mathcal{D}_-(e) = \emptyset$ for some $e \in \mathcal{E}$, then $u_{S^*}(e) = \min\{f_C(T) : T \in \mathcal{D}_-(e)\} - f_C(S^*) = \min\{\emptyset\} = \infty$ (by definition). Similarly, for $\mathcal{D}_+(e) = \emptyset$ we take $l_{S^*}(e) = \infty$.

The following statement can be derived from Theorem 3.3.1. If one excludes an element $e$ from $S^*$, then the objective value of the new problem will be $f_C(S^*)+u_{S^*}(e)$. The same holds for the lower tolerance if the element $e \in \mathcal{E}\setminus S^*$ is included. So a tolerance-based BnB algorithm knows the cost of including or excluding elements before it selects the element to branch on.

## 3.4 Upper Tolerances of the Assignment Problem

In this section, we introduce the tolerance values of the Assignment Problem (AP). Recall that the AP is the problem of assigning $n$ employees to $n$ jobs against minimum costs, given a cost matrix $C$. Each employee is only allowed to perform one job. The assignment of employee $i$ to job $j$ is denoted by the arc $(i, j)$ and has cost $c(i, j)$. An *instance* of the AP is defined by its cost matrix $C$. Let $\mathcal{A}$ denote the set of feasible solutions of the AP instance $C$, and let $\mathcal{A}^*$ be the set of optimal solutions of $C$. We denote the cost of the assignment $A \in \mathcal{A}$ of instance $C$ by $f_C(A)$. The cost is obtained by adding the costs of all arcs $e \in A$, so $f_C(A) = \sum_{e\in A} c(e)$. The solution of the AP can be represented as a set of cycles, and the AP solution is also called a *minimum cycle cover*.

Consider the AP instance with the following cost matrix $C$, borrowed from Balas and Toth (1985).

| City | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | $\infty$ | 2 | 11 | 10 | 8 | 7 | 6 | 5 |
| 2 | 6 | $\infty$ | 1 | 8 | 8 | 4 | 6 | 7 |
| 3 | 5 | 12 | $\infty$ | 11 | 8 | 12 | 3 | 11 |
| 4 | 11 | 9 | 10 | $\infty$ | 1 | 9 | 8 | 10 |
| 5 | 11 | 11 | 9 | 4 | $\infty$ | 2 | 10 | 9 |
| 6 | 12 | 8 | 5 | 2 | 11 | $\infty$ | 11 | 9 |
| 7 | 10 | 11 | 12 | 10 | 9 | 12 | $\infty$ | 3 |
| 8 | 10 | 10 | 10 | 10 | 6 | 3 | 1 | $\infty$ |

The (unique) optimal AP solution $A^*$ consists of the three cycles $K_1 = \{(1,2),(2,3),(3,1)\}$, $K_2 = \{(4,5),(5,6),(6,4)\}$, and $K_3 = \{(7,8),(8,7)\}$. The cost value of this solution is 17.

Take any $A^* \in \mathcal{A}^*$, while $A^*$ need not be unique. The upper tolerance value of any arc $e$ is the maximum increase in the cost $c(e)$ such that $A^*$ remains optimal. More formally, let $f_C(A^*)$ denote the cost of any assignment solution $A^*$ of the instance $C$, and let $C_{\alpha,e}$ denote the instance in which the cost value of arc $e$ is increased with $\alpha$ in comparison with the instance $C$, and the costs of all other arcs remain unchanged. The upper tolerance value of arc $e$ with respect to $A^*$ is denoted by and defined as:

$$u_{A^*}(e) = \max\{\alpha \geq 0 \,:\, A^* \in \arg\min\{f_{C_{\alpha,e}}(A) \,:\, A \in \mathcal{A}^*\}\},$$

Upper tolerances are defined with respect to a fixed optimal solution $A^*$, because $A^*$ need not be unique. Consider an AP instance with two optimal solutions $A_1^*$ and $A_2^*$, and let arc $e \in A_1^* \setminus A_2^*$. Then, by definition, $u_{A_1^*}(e) = 0$, whereas $u_{A_2^*}(e) = \infty$.

Section 3.3 show that the upper tolerance value of the arc $e$ corresponds to the value of the cheapest solution without $e$; the proof is based on Libura (1991). More formally, let $C_{\infty,e}$ be the instance from which $e$ is excluded, let $\mathcal{A}^*_-(e)$ be the set of optimal solutions of $C_{\infty,e}$, and let $A^*_-(e) \in \mathcal{A}^*_-(e)$. The instance $C_{\infty,e}$ is formed by setting the cost value of $e$ to a very large number. The cost of the optimal solution $A^*_-(e)$ of the new instance is $f_C(A^*_-(e))$. We may write $C$ instead of $C_{\infty,e}$, since $e \notin A^*_-(e)$. The upper tolerance of $e$ satisfies $u_{A^*}(e) = f_C[A^*_-(e)] - f_C[A^*]$. In order to compute one upper tolerance value, the additional AP instance $C_{\infty,e}$ needs to be solved.

Consider the AP example above. The upper tolerance of the arc $(7,8)$ is obtained by setting the entry $c(7,8)$ to $\infty$ and solving the newly obtained in-

stance. The optimal solution of the new instance contains the arcs $(1, 8)$ and $(7, 1)$ instead of $(7, 8)$ and $(1, 2)$. The cost of this solution $f_C[A^*_-(7, 8)] = 28$. So the upper tolerance value satisfies: $u_{A^*}[(7, 8)] = f_C[A^*_-(7, 8)] - f_C(A^*) = 28 - 17 = 11$.

Although solving an AP from scratch takes $O(n^3)$ time, it is well known (see, e.g., Balas and Toth (1985)) that for finding an optimal solution $A^*_-(e)$ based on the given AP solution $A^*$, only one labelling procedure in the Hungarian method needs to be performed, which can be done in $O(n^2)$ time.

## 3.5 Survival Sets

This section explores the *branching* step of BnB algorithms. The goal of branching is to find a good or even optimal solution in the fastest possible way. BnB methods generate sequences of steps in which parts of the solution at hand are included and excluded, until an optimal solution of the original problem is found. If a BnB algorithm predicts correctly which element to delete or to insert, then its search tree will be small. So it is important to predict survival sets accurately. Most algorithms base the predictions on cost values, but the question is: do predictions improve if they are based on upper tolerance values, and: how many survival arcs are there on average?

We have selected instances from the ATSPLIB (see Reinelt (1991)) of size $n = 34$ to 171. In our experiments we consider instances with varying degree of symmetry and degree of sparsity. The *degree of symmetry* is defined as the fraction of off-diagonal entries of the cost matrix $\{c_{ij}\}$ that satisfy $c_{ij} = c_{ji}$; the *degree of sparsity* is the percentage of arcs that is missing in an instance. The random instances have degree of symmetry 0, 0.33, 0.66, and 1. The sparse random instances have degree of sparsity of 50%, 75%, and 90%. All randomly generated instances have size 60, 70, and 80. Finally, the instances by Buriol *et al.* (2004) are derived from symmetric TSPLIB instances. The almost symmetric Buriol instances are derived from symmetric instances from the TSPLIB as follows; see Buriol *et al.* (2004). Let $\sigma$ be the average of all distances of the STSPLIB instance, and let $k'$ be a user-defined number. Each intercity distance on the lower diagonal of the cost matrix $C$ of the original symmetric instance is increased by a factor $k\sigma$, where $k$ is randomly drawn from the uniform distribu-

**Table 3.1.** Fraction of survival arcs in optimal AP and ATSP solutions

| Instance type | Fraction of survival arcs |
|---|---|
| ATSPLIB | 53.52% |
| Degree of symmetry 0.33 | 69.29% |
| Degree of symmetry 0.66 | 51.10% |
| Full symmetry | 43.44% |
| Asymmetric random | 80.49% |
| Degree of sparsity 50% | 86.27% |
| Degree of sparsity 75% | 84.23% |
| Degree of sparsity 90% | 83.46% |
| Buriol, $k' = 5$ | 46.72% |
| Buriol, $k' = 50$ | 72.95% |

tion supported on $\{0, \ldots, k'\}$. So the value of $k'$ is a measure of the deviation of the instances from full symmetry.

Table 3.1 shows that the average percentage of common arcs in corresponding AP and ATSP solutions varies between 40 and 80%. Similar investigations show that the Minimum 1-Trees and optimal STSP tours have between 70% and 80% of the edges in common (Helsgaun, 2000).

Let $\mathcal{H}$ denote the set of all feasible tours of an ATSP instance, and define $\mathcal{H}^*$ as the set of optimal tours. Note that $\mathcal{H} \subseteq \mathcal{A}$. Assume that we start with a fixed AP solution $A^* \in \mathcal{A}^*$, and that $H^* \in \mathcal{H}^*$ is a fixed shortest complete tour of the same instance. We explore whether there are relationships between the cost values and the upper tolerance values of arcs and their appearance in $H^*$. These relationships are measured with the *adjusted Rand index*, which measures the relationship between two partitions; see Hubert and Arabie (1985), and with *logistic regression* (Gessner *et al.*, 1988). The costs and the upper tolerances are continuous variables, and are both compared with the partition of $A^*$ into survival and extinction arcs.

In the adjusted Rand index analysis (Hubert and Arabie, 1985), we create partitions based on upper tolerances and costs. First, the arcs in a fixed optimal AP solution $A^*$ are partitioned into two subsets: $IN_1$ contains the survival arcs, and the subset $IN_0$ contains the extinction arcs. Call this partition $IN = \{IN_0, IN_1\}$. We try to replicate $IN$ with partitions $C$ and $U$ based on the costs and tolerance values of the arcs, respectively. Define $C := \{C_0, C_1\}$,

**Table 3.2.** Quality of the predictions using upper tolerances and costs

| | Adjusted Rand indices | | $R^2$ of logit model | |
|---|---|---|---|---|
| Instance type | Tolerance | Cost | Tolerance | Cost |
| ATSPLIB | 0.113 | -0.003 | 0.112 | 0.035 |
| Degree of symmetry 0.33 | 0.152 | 0.007 | 0.169 | 0.017 |
| Degree of symmetry 0.66 | 0.188 | 0.028 | 0.132 | 0.015 |
| Full symmetry | 0.158 | 0.013 | 0.007 | 0.011 |
| Asymmetric random | 0.287 | 0.039 | 0.299 | 0.023 |
| Sparsity 50% | 0.361 | 0.017 | 0.407 | 0.015 |
| Sparsity 75% | 0.252 | 0.033 | 0.382 | 0.013 |
| Sparsity 90% | 0.219 | 0.032 | 0.342 | 0.017 |
| Buriol, $k' = 5$ | -0.019 | -0.023 | 0.010 | 0.006 |
| Buriol, $k' = 50$ | 0.217 | 0.0303 | 0.172 | 0.028 |

where $C_0 = \{e \in A^* : c(e) \geq c^*\}$, $C_1 := \{e \in A^* : c(e) < c^*\}$, and determine $c^*$ in such a way that $|C_0| = |IN_0|$. Arcs are partitioned into a set of low cost arcs $C_1$ and a class of high cost arcs $C_0$. If it is true that all high cost arcs are not in the given shortest tour, then the sets $IN_0$ and $C_0$ and the sets $IN_1$ and $C_1$ coincide and cost values lead to a perfect prediction. Similarly, define $U = \{U_0, U_1\}$, where $U_0 := \{e \in A^* : u_{A^*}(e) < u^*\}$, $U_1 := \{e \in A^* : u_{A^*}(e) \geq u^*\}$, and determine $u^*$ in such a way that $|U_0| = |IN_0|$.

The *adjusted Rand index* measures how similar the each of both partitions $U$ and $C$ are in comparison with the ideal partition into survival and extinction arcs $IN$. The formula of the adjusted Rand index is given in Section 5.4. The more similar two partitions are, the higher the adjusted Rand index between both partitions is. An adjusted Rand index of 1 indicates that for each nonempty class $A_i$ of partition $A$, there exists a class $B_j$ of partition $B$ such that $A_i = B_j$. The expected adjusted Rand index is 0, if both partitions assign objects to classes randomly having the original number of objects in each class (Hubert and Arabie, 1985).

The adjusted Rand indices between $IN$ and $C$ and between $IN$ and $U$ are shown in Table 3.2. They are larger for the tolerance-based partitions $U$ than for the cost-based partitions $C$, which indicates that predictions are better if they are based on upper tolerance values.

The adjusted Rand index analysis makes splits in the data based on the cost

and the upper tolerance values. It does not include the distances in cost or tolerance value of an arc from the split value. The following problem arises. Suppose that an arc with a very high upper tolerance value is not in any shortest ATSP tour. Such an arc is very likely to be excluded already in an early stage of a tolerance-based branching process. If this event occurs, the predictions of upper tolerances should get a bad rating for this instance. The same holds for arcs with low cost values which are not in a shortest ATSP tour. We define the binary variable $IN$, ranging over all arcs $e \in A^*$, as follows: $IN(e) = 1$ if $e \in H^*$ and $IN(e) = 0$ otherwise.

An appropriate method for estimating the relationship between a dependent binary variable and independent continuous variables is *logistic regression* (Hair *et al.*, 1998). Logistic regression is usually applied to explain or predict choices in choice modeling, for example the choice between buying and not buying a product (Hosmer and Lemeshow, 1989). Based on the values of the independent variables, probabilities $\pi(e)$ are estimated of the event that the independent variable attains the value 1 for the observation $e$. The fit of a model is good if these probabilities $\pi(e)$ are close to the actual observed values of the dependent variable.

A general measure to determine the fit of a logistic regression model, also called *logit model*, is the $R^2$ *for a logit model* $R^2_{logit}$ (Gessner *et al.*, 1988), which compares the predictive power of a logit model to the predictive power of a model without independent variables. If $R^2_{logit} = 1$, then all the variance in the independent variable is explained and predictions are perfect. On the other hand, if $R^2_{logit} = 0$, the independent variables in the model offer no information about the dependent variable. $R^2_{logit}$ is similar to $R^2$ in linear regression.

In order to analyze survival sets, we construct for each instance two separate logit models. In the cost-based model the dependent variable $IN$ is explained by the cost values of the arcs in an assignment solution; the independent variable in the tolerance-based model is formed by the upper tolerance values. The values of $R^2_{logit}$ of both models are presented in Table 3.2.

The values of $R^2_{logit}$ are higher for the tolerance-based models, except for fully symmetric instances. These results confirm that predictions based on upper tolerance values are clearly better than predictions based on costs.

## 3.6 Tolerance-Based Lower Bounds for the ATSP

In this section, we use the upper tolerances of an optimal AP solution to construct tight lower bounds for the corresponding ATSP. These lower bounds are introduced in Goldengorin *et al.* (2004). Recall that, if the lower bound of a subproblem is increased, then this subproblem is more likely to be fathomed during the execution of the BnB algorithm.

In BnB algorithms, lower bounds can be obtained by removing sources of infeasibility with respect to the original problem from the current solution. We call such sources of infeasibility *offenders*. In case of the ATSP and its AP relaxation, offenders are subcycles. Let $A^*$ consist of $k(>1)$ cycles, say, $A^* = \cup_{i=1}^{k} K_i$. We define $\mathcal{C}(A^*)$ as the set of all cycles in $A^*$, so $\mathcal{C}(A^*) = \{K_1, \ldots, K_k\}$.

In order to "break" a subcycle $K$ (meaning that this cycle does not appear in subsequent AP solutions), at least one arc must be removed. Recall that the cost of removing an arc is equal to its upper tolerance value. Hence, the minimum cost of breaking a subcycle is equal to the lowest upper tolerance value in that cycle. We denote and define for each $K \in \mathcal{C}(A^*)$ this value by $u_{A^*}^K := \min\{u_{A^*}(e) : e \in K\}$. Theorem 3.6.1 shows that the cost of breaking a cycle by deleting a minimum tolerance arc can be used to increase the lower bound.

**Theorem 3.6.1.** *Let $A^*$ and $H^*$ be optimal solutions to AP and ATSP instances, respectively, with the same cost matrix $C$. Assume that $A^*$ consists of at least two cycles. Then for each $K \in \mathcal{C}(A^*)$, the following inequalities hold:*

$$f_C(A^*) \leq f_C(A^*) + u_{A^*}^K \leq f_C(H^*).$$

*Proof.* The first inequality is obvious, since $u_{A^*}(e) \geq 0$ for every $e \in A^*$. Now we show that $f_C(A^*) + u_{A^*}^K \leq f_C(H^*)$. Take any $K \in \mathcal{C}(A^*)$, and take any $e \in K \setminus H^*$. Let $\mathcal{A}_-(e)$ be the set of all AP solutions without $e$, so $\{A \in \mathcal{A} : e \notin A\}$. Moreover, let $A_-^*(e) \in \arg\min\{f_C(A) : A \in \mathcal{A}_-(e)\}$. From Section 3.3, we have that $u_{A^*}(e) = f_C[A_-^*(e)] - f_C(A^*)$. Since $H^* \in \mathcal{A}_-(e)$ $f_C(A^*) + u_{A^*}(e) = f_C[A_-^*(e)] \leq f_C(H^*)$. Hence $f_C(A^*) + u_{A^*}^K \leq f_C(A^*) + u_{A^*}(e) = f_C[A_-^*(e)] \leq f_C(H^*)$. □

Based on Theorem 3.6.1, we may ask the question which subcycle in $A^*$ should be "broken". The most effective choice is the cycle $K$ in which $u_{A^*}^K$ is

maximal, since it causes the largest increase in the lower bound $f_C(A^*) + u^K_{A^*}$. However, all tolerance values in all cycles must be computed to guarantee that we obtain an arc with maximal value of $u^K_{A^*}$. This is usually a time-consuming matter. Hence, it may be worthwhile to restrict the tolerance calculations to a 'not too large' subset of $\mathcal{C}(A^*)$. Let $O \subseteq \mathcal{C}(A^*)$. Denote and define the *bottleneck tolerance* with respect to $O \subseteq \mathcal{C}(A^*)$ by $bu_{A^*}(O) := \max\{u^K_{A^*} : K \in O\}$, and the corresponding *bottleneck bound* by $lb_{A^*}(O) = f_C(A^*) + bu_{A^*}(O)$. The choice for the set of offenders $O$ determines the values of the bottleneck tolerances and bounds. For example, if $\mathcal{C}(A^*) = \{K_1, K_2\}$, $u^{K_1} = 1$, and $u^{K_2} = 5$, then $bu_{A^*}(\mathcal{C}(A^*)) = bu_{A^*}(\{K_1\}) = 5$, whereas $bu_{A^*}(\{K_2\}) = 1$ and $bu_{A^*}(\emptyset) = 0$.

We consider special sets of offenders in $\mathcal{C}(A^*)$:

The *Entire Cycle Set (ECS)*. Define $O_E := \mathcal{C}(A^*)$. So the bottleneck tolerance value in the entire set of subcycles of $A^*$ is taken. Note that $bu_{A^*}(O_E) \geq bu_{A^*}(O)$ for every $O \subseteq \mathcal{C}(A^*)$ and for every $A^*$. This lower bound corresponds with the Exact Bottleneck Bound from Goldengorin *et al.* (2004).

The *Smallest Cycle Set (SCS)*. Define $O_S := \{K^*\}$, where $K^*$ is a cycle of $A^*$ with the smallest cardinality, i.e., $K^* \in \arg\min\{|K| : K \in \mathcal{C}(A^*)\}$. This lower bound corresponds to the Approximate Bottleneck Bound from Goldengorin *et al.* (2004).

The concept of bottleneck tolerances uses the structure of an assignment solution to increase the lower bound. For instance, suppose that an assignment solution of a randomly generated instance consists of many small cycles. We may expect a high bottleneck tolerance value, since the maximum is taken from a large set of numbers. Note also that if an assignment consists of a large number of cycles, then, on average, the gap between the AP and the ATSP solution values is wide. So there is a relationship between the size of the gap and the value of the bottleneck tolerance.

The ECS choice leads to the tightest upper tolerance-based lower bounds. The SCS choice is taken into account, because it gives a good approximation for the ECS bound in a short time. We claim that, in many situations, the value of $bu_{A^*}(O_E)$ is attained on a smallest cycle, and hence, $bu_{A^*}(O_S)$ is a good approximation of it. The intuition behind this claim is the following. Suppose

upper tolerance values are randomly dispersed over the arcs of a minimum cycle cover. The minimum tolerance value of a small cardinality cycle is then relatively large; therefore, it is likely that $bu_{A^*}(O_E)$ is attained on a smallest cycle of $A^*$. Table 3.3 shows that the fraction of subproblems in a BnB search tree for which this event occurs, is about 45%.

**Table 3.3.** Percentage $bu_{A^*}(O_E)$ in smallest cycles and reductions by ECS and SCS lower bounds

| Instance | $bu_{A^*}(O_E)$ in smallest cycles | $r(O_E)$ | $r(O_S)$ |
|---|---|---|---|
| ATSPLIB | 46.38% | 19.97% | 6.39% |
| Degree of symmetry 0.33 | 60.98% | 34.62% | 17.07% |
| Degree of symmetry 0.66 | 69.48% | 26.66% | 12.27% |
| Full symmetry | 88.49% | 21.64% | 14.61% |
| Asymmetric random | 43.09% | 50.47% | 43.31% |
| Degree of sparsity 50% | 43.73% | 56.50% | 48.99% |
| Degree of sparsity 75% | 44.06% | 45.78% | 40.45% |
| Degree of sparsity 90% | 44.49% | 49.86% | 35.45% |

The next natural question is: what is the difference in quality between $lb_{A^*}(O_C)$ and $lb_{A^*}(O_S)$? To measure the quality of a lower bound, we introduce the *reductions* $r(O)$ of the gap between $f_C(A^*)$ and $f_C(H^*)$ achieved by the lower bound $lb_{A^*}(O)$. Define $r(O) = \frac{bu_{A^*}(O)}{f_C(H^*)-f_C(A^*)} \times 100\%$. Table 3.3 compares $r(O_E)$ and $r(O_S)$. The results show that, for (quasi-)symmetric and ATSPLIB instances, the ECS choice clearly constructs better lower bounds than the SCS choice. However, the SCS choice gives a satisfactory approximation for asymmetric random and sparse instances, while it is generally much faster to compute.

In BnB settings, lower bounds can be computed at every node of the search tree. A high quality bound, such as $lb_{A^*}(O_E)$ allows the BnB algorithm to discard a large number of nodes, but it requires long computing times at each node considered. In order to find the balance between bound quality and computing times, computational experiments are performed in Section 3.9.

The approaches are clarified with the ATSP example from Section 3.4. Recall that the unique AP solution $A^*$ consists of the three cycles $K_1 = \{(1,2),(2,3),$

$(3,1)\}$, $K_2 = \{(4,5),(5,6),(6,4)\}$, and $K_3 = \{(7,8),(8,7)\}$. Moreover, $f_C(A^*) = 17$ and $f_C(H^*) = 26$. Figure 3.1 depicts $A^*$ and the upper tolerance values of the arcs.

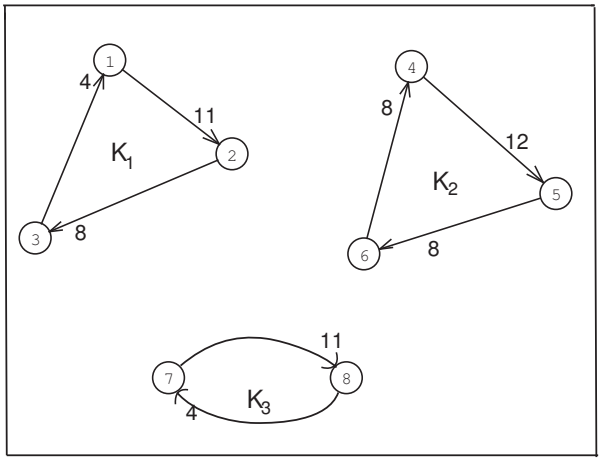


**Figure 3.1.** Minimum cycle cover with arc upper tolerances

In this example, $u^{K_1} = 4$, $u^{K_2} = 8$, and $u^{K_3} = 4$. So $bu_{A^*}(O_E) = 8$ is attained on the arcs $(5,6)$ and $(5,6)$ in cycle $K_2$, and $bu_{A^*}(O_S) = 4$ on arc $(8,7)$ in the smallest cycle $K_3$. Therefore, $lb_{A^*}(O_S) = 21$ and $lb_{A^*}(O_E) = 25$. Since $f_C(H^*) = 26$, $r(O_E) = \frac{8}{26-17} \times 100\% = 88.8\%$, $r(O_S) = \frac{4}{26-17} \times 100\% = 44.4\%$. For this instance, $lb_{A^*}(O_E) = 25$ is tighter than all other bounds discussed in Balas and Toth (1985).

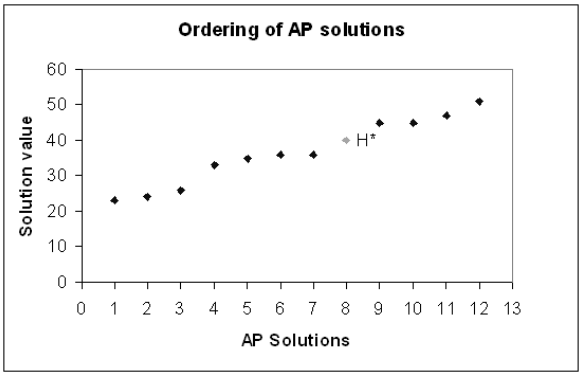


**Figure 3.2.** Enumeration of AP solutions

Applying the concept of bottleneck tolerances not only increases lower

bounds, but it also strengthens the branching. The previous section shows that it is worthwhile to branch on an arc with a small upper tolerance value. But which one should we choose? Figure 3.2 depicts an enumeration of feasible solutions of an AP instance in a non-decreasing order of cost values. A careful branching strategy, which branches on a smallest upper tolerance arc, obtains all AP solutions with cost smaller than $f_C(H^*)$. However, if an algorithm branches on a bottleneck tolerance arc, then it traverses the enumeration with larger steps, and it is likely to arrive at $H^*$ in fewer branching steps. That is, of course, if it does not exclude survival arcs. Table 3.3 indicates that the exclusion of an ECS bottleneck arc from an asymmetric, randomly generated instance brings the solution value of the next subproblem on average about 50% closer to $f_C(H^*)$. We propose branching rules based on bottleneck tolerances arcs obtained by the ECS and the SCS choices. We call these the *ECS* and *SCS branching rules*, respectively.

## 3.7 The algorithms

In the previous sections, have discussed tolerance-based branching rules and lower bounds for the ATSP. In Goldengorin *et al.* (2004), it is shown that search tree reductions of tolerance-based DFS algorithms are the largest when the branching rule and the lower bound act in conjunction. We explain this in the next section with the so-called *synergy effect*. Two tolerance-based algorithms are considered in the experiments: $BnB(ECS)$ uses the ECS branching rule and lower bound, whereas $BnB(SCS)$ uses the SCS branching rule and lower bound. whereas $BnB(SCS)$ uses the SCS branching rule and lower bound. After computing an assignment solution, the upper tolerance values of arcs in the specified cycle set are determined for the lower bound. When the subproblem cannot be discarded, these upper tolerance values are used for branching. The cost-based benchmark $BnB(Cost)$ is a DFS algorithm which branches on the longest arc in a smallest cycle of the current AP solution. The algorithms apply the subtour elimination scheme from Carpaneto and Toth (1980), and the reduction procedure from Carpaneto *et al.* (1995) is used at the top node of the search tree to determine which arcs will never appear in an optimal solution. These arcs are then removed from the arc set. Moreover, the algorithms apply the solver from

Jonker and Volgenant (1986) to solve the APs and to compute the upper tolerance values. These algorithms are proved to be competitive in the AP solver comparison by Dell'Amico and Toth (2000).

The flowchart in Figure 3.3 provides a schematic view of a tolerance-based BnB algorithm. In each subproblem of $BnB(ECS)$, $n$ upper tolerances are computed where $n$ is the dimension of the current subproblem. $BnB(SCS)$ uses the SCS branching rule and lower bound. Hence, the complexities are $O(n^3)$ and $O(|K^*|n^2)$, respectively. The ECS branching rule selects a subcycle in which the largest minimum upper tolerance value is achieved. Subsequently, it branches on the arcs in this cycle in non-decreasing order of tolerances.

The performance of our DFS methods is compared to the performance of the CDT algorithm by Carpaneto *et al.* (1995). In Fischetti *et al.* (2002), the CDT code is not able to solve the ATSPLIB instances ft53, ftv170, kro124p, p43, and ry48p within the specified time limit of 1,000 seconds. Fischetti *et al.* (2002) also note that the CDT approach "either solve the problems within CPU 1,000 seconds, or the final gap is too large to hope in a convergence within 10,000 seconds". By setting the maximum computing time to 10,000 seconds, the search trees are kept reasonably small. Our Pentium 4 computer is approximately twice as fast as the Alpha 500 MHz used for the experiments in Fischetti *et al.* (2002); see the machine speed comparison from Johnson (2006). The BnB algorithms are listed in Table 3.4.

**Table 3.4.** BnB algorithms

| Algorithm | Description |
|---|---|
| $CDT$ | BFS algorithm by Carpaneto *et al.* (1995) |
| $BnB(Cost)$ | Cost-based DFS algorithm |
| $BnB(ECS)$ | Algorithm with ECS lower bound and branching rule |
| $BnB(SCS)$ | Algorithm with SCS lower bound and branching rule |

## 3.8 BnB with Tolerance-Based Branching Rules and Lower Bounds

The BnB search trees with tolerance-based lower bounds, with tolerances based branching rules, and with a combination of both are compared in Goldengorin

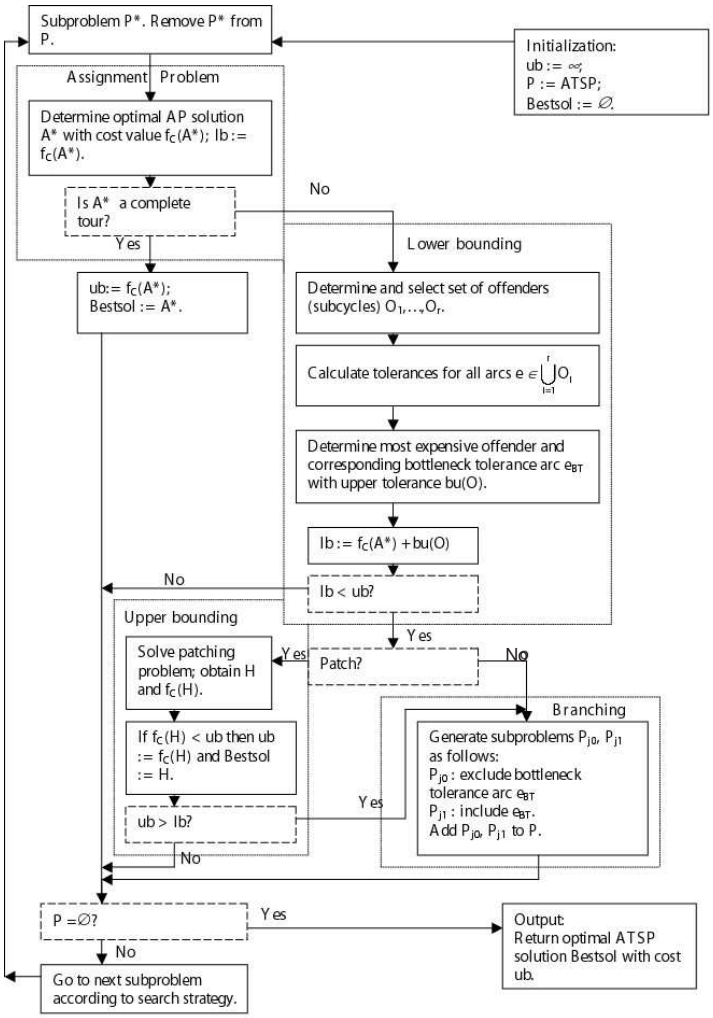


**Figure 3.3.** Flowchart of a tolerance-based BnB algorithm

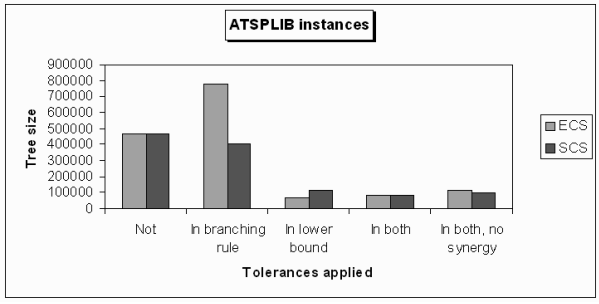


**Figure 3.4.** Synergy effects for ATSPLIB instances

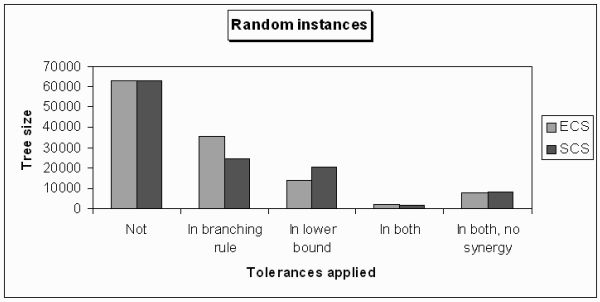


**Figure 3.5.** Synergy effects for random instances

*et al.* (2004). They show that BnB algorithms with only new lower bounds have smaller search trees than the conventional algorithm, $BnB(Cost)$. The SCS branching rules also achieves reductions for most instances, but the ECS branching rule only reduces the search trees for random instances. The reductions of the joint use of tolerance-based lower bounds and branching rules are often larger than the reductions when they are used separately.

For many instances, branching on an SCS arc turns out to be more effective than branching on an ECS arc. This is counterintuitive, but the discussion in Section 3.5 may explain this phenomenon. It was observed there that small upper tolerance arcs are more likely to be in an optimal ATSP solution than large upper tolerance arcs. Since the upper tolerance value of an ECS arc is generally higher than the upper tolerance value of an SCS arc, the ECS branching rule is more likely to delete survival arcs than the SCS branching rule. It may also explain why the ECS branching rule leads to larger search trees than the cost-based branching rule in Figure 3.4.

**Table 3.5.** Search tree sizes with tolerance-based branching rules (BR) and lower bounds (LB) for ATSPLIB instances

| | | | Entire Cycle Set | | | | Smallest Cycle Set | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | $n$ | $BnB(Cost)$ | BR | LB | $BnB(ECS)$ | $\frac{BnB(Cost)}{BnB(ECS)}$ | BR | LB | $BnB(SCS)$ | $\frac{BnB(Cost)}{BnB(SCS)}$ |
| ft53 | 53 | 20111 | $*$ | 7039 | 336586 | | 89511 | 17703 | 20545 | 0.98 |
| ft70 | 70 | 25831 | 70047 | 5619 | 5861 | 4.41 | 22843 | 6717 | 4993 | 5.17 |
| ftv33 | 34 | 7065 | 3867 | 1983 | 748 | 9.45 | 6007 | 3137 | 1569 | 4.50 |
| ftv35 | 36 | 6945 | 18871 | 2553 | 3109 | 2.23 | 12047 | 3219 | 2265 | 3.07 |
| ftv38 | 39 | 6195 | 9726 | 1381 | 1523 | 4.07 | 14663 | 2821 | 2387 | 2.60 |
| ftv44 | 45 | 619 | 976 | 187 | 165 | 3.75 | 937 | 249 | 195 | 3.17 |
| ftv47 | 48 | 29025 | 29121 | 8017 | 3302 | 8.79 | 48345 | 9703 | 8393 | 3.46 |
| ftv55 | 56 | 92447 | 98433 | 12413 | 10100 | 9.15 | 114641 | 26483 | 26023 | 3.55 |
| ftv64 | 65 | 43441 | 89752 | 9449 | 8319 | 5.22 | 162639 | 11007 | 17173 | 2.53 |
| ftv70 | 71 | 253873 | 459532 | 25939 | 50024 | 5.07 | 136296 | 52289 | 18937 | 13.41 |

$*$Memory exhausted

**Table 3.6.** Search tree sizes with tolerance-based branching rules (BR) and lower bounds (LB) for random instances

| | | Entire Cycle Set | | | | Smallest Cycle Set | | | |
|---|---|---|---|---|---|---|---|---|---|
| $n$ | $BnB(Cost)$ | BR | LB | $BnB(ECS)$ | $\frac{BnB(Cost)}{BnB(ECS)}$ | BR | LB | $BnB(SCS)$ | $\frac{BnB(Cost)}{BnB(SCS)}$ |
| 60 | 3808 | 3275 | 978 | 323 | 11.79 | 1032 | 2832 | 221 | 17.23 |
| 70 | 4528 | 4085 | 1138 | 312 | 14.51 | 1286 | 1781 | 247 | 18.33 |
| 80 | 9014 | 3937 | 2414 | 217 | 41.53 | 2494 | 2746 | 256 | 35.21 |
| 100 | 9002 | 3645 | 1978 | 174 | 51.74 | 2188 | 5306 | 135 | 66.68 |
| 200 | 36390 | 20497 | 7114 | 858 | 42.41 | 17612 | 7612 | 796 | 45.72 |

The search trees of ATSPLIB and random instances are depicted in Figure 3.4. The first column shows the average search tree of $BnB(Cost)$, the second column the trees with tolerance-based branching rules, the third the trees with tolerance-based lower bounds, and the fourth the values of $BnB(ECS)$ and $BnB(SCS)$. If the reductions of tolerance-based lower bounds and branching rules would have been independent, then the actual sizes of the search tree of $BnB(ECS)$ and $BnB(SCS)$ would be equal to their expected sizes, formed by the reduction of the branching rule times the reduction of the lower bound. These values are represented in the column "Both without synergy" in Figures 3.4 and 3.5. However, the actual trees of $BnB(ECS)$ and $BnB(SCS)$ are lower than the expected search trees, indicating that the joint use of tolerance-based lower bounds and branching rules leads to additional search tree reductions. We call this remarkable phenomenon the *synergy effect*. The algorithms $BnB(ECS)$ and $BnB(SCS)$ benefit from the synergy effect.

## 3.9 Computational Experiments with ATSP Instances

In this section, computational experiments are conducted on the algorithms listed in Table 3.4. The central questions are: do tolerance-based algorithms have smaller search trees, and if this is true, are the reductions sufficient to compensate for the time invested in tolerance computations?

The question we ask now is whether the search tree reductions of $BnB(ECS)$ and $BnB(SCS)$ are sufficient to compensate for the time invested in the tolerance calculations. The cost-based DFS benchmark is $BnB(Cost)$. In order to compare the quality of our DFS methods to BFS methods, we report the solution times of the CDT algorithm on the same computer as well.

We have selected the set of practical ATSPLIB instances by Reinelt (1991). Moreover, we have selected the so-called Buriol instances; the description is given Section 3.5. We solve these practical problems in increasing order of size until the instances encountered cannot be solved within our time limit of 3600 seconds. The random instances have degree of symmetry 0, 0.33, 0.66, and 1, with instance size 60, 70, and 80. The sparse random instances have degree of sparsity of 50%, 75%, and 90%, and their sizes are 100, 200, and 400. For each randomly generated problem set and for all instance sizes, 10 instances

have been generated. The experiments are conducted on a Pentium 4 computer with 256 MB RAM memory and 2 GHz speed. For the ATSPLIB and the Buriol instances, we include three statistics to summarize the results: the number of unsolved instances, the average solution time and the median solution time. The median solution time is included, because the solution times of relatively hard instances are overrepresented and easy instances are underrepresented in the average solution times. The time limit for all algorithms, also $CDT$, is 3600 seconds; when no optimal solution is found, we report 'Not' in the corresponding table and the instance is considered unsolved. The average and median solution times and search tree sizes contain only instances that are solved by all tested BnB algorithms.

**Table 3.7.** Search tree sizes and solution times of small ATSPLIB instances

| | | $CDT$ | $BnB(Cost)$ | | $BnB(ECS)$ | | $BnB(SCS)$ | |
|---|---|---|---|---|---|---|---|---|
| Instance | $n$ | Time | Tree | Time | Tree | Time | Tree | Time |
| br17 | 17 | 3 | 3674829 | 24.67 | 86585 | 4.18 | 1034255 | 22.64 |
| ftv33 | 34 | 0 | 7065 | 0.22 | 2195 | 0.82 | 1362 | 0.11 |
| ftv35 | 36 | 0 | 6945 | 0.22 | 2307 | 0.99 | 1965 | 0.27 |
| ftv38 | 39 | 0 | 6195 | 0.22 | 1381 | 0.82 | 2091 | 0.44 |
| p43 | 43 | Not | Not | 3600.00 | Not | 3600.00 | Not | 3600.00 |
| ftv44 | 45 | 0 | 619 | 0.05 | 453 | 0.44 | 171 | 0.05 |
| ftv47 | 48 | 0 | 29025 | 1.37 | 7752 | 7.20 | 7692 | 1.48 |
| ry48p | 48 | Not | 9178227 | 459.84 | 183511 | 262.03 | 601713 | 105.88 |
| ft53 | 53 | Not | 20111 | 2.53 | 336586 | 448.13 | 19200 | 6.15 |
| ftv55 | 56 | 1 | 92447 | 4.56 | 17490 | 24.12 | 23034 | 9.12 |
| ftv64 | 65 | 1 | 43441 | 3.24 | 9491 | 20.88 | 15509 | 10.77 |
| ft70 | 70 | 1 | 25831 | 4.01 | 4123 | 11.70 | 4756 | 1.87 |
| ftv70 | 71 | 2 | 253873 | 24.07 | 34838 | 112.36 | 17694 | 8.57 |
| ftv90 | 91 | 1 | 7059 | 1.54 | 643 | 6.15 | 4023 | 1.87 |
| kro124p | 100 | Not | Not | 3600.00 | Not | 3600.00 | Not | 3600.00 |
| ftv100 | 101 | 13 | 371385 | 92.20 | 1643 | 21.87 | 54688 | 30.93 |
| ftv110 | 111 | 3 | 583133 | 160.44 | 6313 | 112.03 | 79045 | 89.56 |
| ftv120 | 121 | 31 | 3892497 | 1088.57 | 13721 | 305.71 | 137563 | 327.31 |
| ftv130 | 131 | 32 | 256855 | 95.71 | 715 | 19.01 | 11269 | 28.52 |
| ftv140 | 141 | 8 | 78951 | 42.25 | 1683 | 55.82 | 12667 | 13.30 |
| ftv150 | 151 | 114 | 15437 | 9.18 | 625 | 26.37 | 3635 | 4.67 |
| ftv160 | 161 | 72 | Not | 3600.00 | Not | 3600.00 | 330458 | 496.92 |
| ftv170 | 171 | 2321 | 1796149 | 1299.34 | Not | 3600.00 | 412059 | 656.59 |
| rbg323 | 323 | 0 | 3 | 0.00 | 1 | 0.49 | 1 | 0.05 |
| rbg358 | 358 | 0 | 3 | 0.05 | 1 | 0.55 | 1 | 0.00 |
| rbg403 | 403 | 0 | 3 | 0.05 | 1 | 0.82 | 1 | 0.05 |
| rbg443 | 443 | 0 | 3 | 0.11 | 3 | 3.02 | 2 | 0.05 |
| Unsolved | | 4 | | 3 | | 4 | | 2 |
| Average | | 10.00 | 445028 | 73.94 | 9141 | 35.02 | 67211 | 26.27 |
| Median | | 1.00 | 25831 | 3.24 | 1683 | 7.20 | 4756 | 1.87 |

Firstly, we compare our DFS algorithms with the CDT benchmark in Tables 3.7, 3.8, and 3.9. Although the median solution times of CDT are generally

**Table 3.8.** Average search tree sizes and solution times of asymmetric random instances

| | CDT | $BnB(Cost)$ | | $BnB(ECS)$ | | $BnB(SCS)$ | |
|---|---|---|---|---|---|---|---|
| $n$ | Time | Tree | Time | Tree | Time | Tree | Time |
| 60 | 0.00 | 380.8 | 0.03 | 32.3 | 0.12 | 22.1 | 0.02 |
| 70 | 0.00 | 452.8 | 0.04 | 31.2 | 0.18 | 24.7 | 0.03 |
| 80 | 0.10 | 901.4 | 0.13 | 21.7 | 0.24 | 25.6 | 0.07 |
| 100 | 0.10 | 900.2 | 0.19 | 17.4 | 0.26 | 13.5 | 0.05 |
| 200 | 0.10 | 3639.0 | 3.30 | 85.8 | 7.30 | 79.6 | 0.84 |
| 300 | 0.30 | 17849.8 | 48.10 | 93.6 | 28.70 | 150.6 | 5.34 |
| 400 | 0.20 | 28499.4 | 141.00 | 74.2 | 54.10 | 121.6 | 11.38 |
| 500 | 0.40 | 43457.6 | 368.70 | 187.8 | 268.40 | 225.3 | 35.98 |
| 1000 | 1.50 | 92289.0 | 3951.60 | 142.1 | 1556.90 | 373.9 | 157.23 |

**Table 3.9.** Average search tree sizes and solution times of symmetric and sparse instances

| | | CDT | BnB(Cost) | | BnB(ECS) | | BnB(SCS) | |
|---|---|---|---|---|---|---|---|---|
| Instance | $n$ | Time | Tree | Time | Tree | Time | Tree | Time |
| Degree of sparsity 50% | 100-400 | 0.6 | 12291 | 44.7 | 90 | 39.1 | 60 | 2.33 |
| Degree of sparsity 75% | 100-400 | 0.7 | 12882 | 48.9 | 109 | 41.57 | 81 | 3.90 |
| Degree of sparsity 90% | 100-400 | 0.7 | 14109 | 55.63 | 116 | 63.63 | 84 | 4.70 |
| Degree of symmetry 0.33 | 60-80 | 0.1 | 1963 | 0.27 | 195 | 0.86 | 139 | 0.10 |
| Degree of symmetry 0.66 | 60-80 | 0.5 | 6763 | 1.08 | 2193 | 9.63 | 1097 | 0.62 |
| Full symmetry | 60-80 | [1] | 446335 | 58.63 | 30347 | 116.73 | 379412 | 121.03 |

**Table 3.10.** Search tree sizes and solution times of Buriol instances, $k' = 5$

| | | $CDT$ | $BnB(Cost)$ | | $BnB(ECS)$ | | $BnB(SCS)$ | |
|---|---|---|---|---|---|---|---|---|
| Instance | $n$ | Time | Tree | Time | Tree | Time | Tree | Time |
| ulysses16 | 16 | 1 | 10165 | 0.11 | 4773 | 0.27 | 2337 | 0.05 |
| ulysses22 | 22 | 45 | 857027 | 10.27 | 209765 | 24.56 | 189735 | 7.36 |
| gr24 | 24 | 0 | 1329 | 0.05 | 401 | 0.16 | 371 | 0.05 |
| fri26 | 26 | 0 | 12725 | 0.22 | 697 | 0.27 | 1943 | 0.11 |
| bayg29 | 29 | 3600 | 6555 | 0.16 | 29939 | 49.34 | 5317 | 0.33 |
| bays29 | 29 | 0 | 13931 | 0.33 | 2037 | 0.66 | 4346 | 0.33 |
| gr48 | 48 | 0 | 3331593 | 171.21 | 155 | 0.00 | 196635 | 25.38 |
| att48 | 48 | 1252 | 63782833 | 2815.93 | 439781 | 543.19 | 120297 | 16.92 |
| eil51 | 51 | 1 | 1193015 | 66.54 | 95403 | 138.63 | 10167 | 1.37 |
| pr76 | 76 | 3600 | Not | 3600.00 | Not | 3600.11 | Not | 3600.05 |
| eil76 | 76 | 3600 | 8772639 | 1058.55 | 9013 | 48.41 | 652530 | 210.93 |
| gr96 | 96 | 3600 | Not | 3600.00 | Not | 3600.05 | Not | 3600.05 |
| Unsolved | | 4 | | 2 | | 2 | | 2 |
| Average | | 162.38 | 8650327 | 383.08 | 94127 | 88.47 | 65729 | 6.45 |
| Median | | 0.50 | 435479 | 5.30 | 3405 | 0.47 | 7257 | 0.85 |

**Table 3.11.** Search tree sizes and solution times of Buriol instances, $k' = 50$

| | | $CDT$ | $BnB(Cost)$ | | $BnB(ECS)$ | | $BnB(SCS)$ | |
|---|---|---|---|---|---|---|---|---|
| Instance | $n$ | Time | Tree | Time | Tree | Time | Tree | Time |
| ulysses16 | 16 | 0 | 2481 | 0.00 | 135 | 0.00 | 181 | 0.00 |
| ulysses22 | 22 | 0 | 35577 | 0.38 | 4270 | 0.71 | 4845 | 0.22 |
| att48 | 48 | 0 | 147201 | 6.65 | 19 | 0.50 | 37 | 0.00 |
| bayg29 | 29 | 0 | 177 | 0.00 | 7 | 0.00 | 7 | 0.00 |
| bays29 | 29 | 0 | 323 | 0.00 | 153 | 0.50 | 23 | 0.00 |
| eil51 | 51 | 1 | 16869 | 0.93 | 9 | 0.50 | 13 | 0.00 |
| fri26 | 26 | 0 | 2481 | 0.00 | 1014 | 0.33 | 33 | 0.00 |
| gr24 | 24 | 0 | 49 | 0.00 | 9 | 0.00 | 19 | 0.00 |
| gr48 | 48 | 0 | 4503 | 0.27 | 43 | 0.50 | 265 | 0.50 |
| pr76 | 76 | 0 | 1643 | 0.27 | 424 | 3.80 | 91 | 0.50 |
| eil76 | 76 | 0 | 243 | 0.00 | 9 | 0.50 | 13 | 0.00 |
| gr96 | 96 | 0 | 851473 | 218.19 | 59613 | 763.90 | 403 | 0.44 |
| kroA100 | 100 | 0 | 21282 | 38.19 | 95 | 1.32 | 82 | 0.05 |
| kroD100 | 100 | 0 | 21294 | 0.11 | 201 | 2.36 | 98 | 0.05 |
| rd100 | 100 | 1 | 7910 | 38.02 | 25 | 0.33 | 434 | 0.22 |
| eil101 | 101 | 0 | 629 | 0.05 | 3 | 0.05 | 21 | 0.00 |
| lin105 | 105 | 0 | 14379 | 42.47 | 17 | 0.27 | 15 | 0.00 |
| ch130 | 130 | 3600 | Not | 3600.00 | 38221 | 969.73 | Not | 3600.00 |
| ch150 | 150 | 3600 | Not | 3600.00 | Not | 3600.00 | Not | 3600.00 |
| Unsolved | | 2 | | 2 | | 1 | | 2 |
| Average | | 0.12 | 66383 | 20.33 | 3885 | 45.62 | 387 | 0.12 |
| Median | | 0.00 | 4503 | 0.27 | 69 | 0.50 | 37 | 0.00 |

the shortest for the ATSPLIB and the Buriol instances, CDT solves the smallest number instances. The CDT algorithm solves randomly generated and small ATSPLIB instances in very short solution times compared to our DFS algorithms. However, the CDT algorithm has more difficulties with the harder and larger ATSPLIB instances, and with symmetric and Buriol instances. The poor performance of the CDT algorithm for difficult problem instances can be explained as follows. Our DFS algorithms use the AP solution of the parent subproblem to speed up the solution of the current subproblem. A BnB algorithm with BFS strategy, on the other hand, proceeds more or less randomly through the search tree, so that it is too memory-intensive to use the AP solution of the parent subproblem. Carpaneto *et al.* (1995) try to overcome this problem by using the AP solution of the top node of the search tree as a starting solution. The AP solutions in the beginning of the BnB process are computed quickly, in $O(n^2)$ time, since these APs are similar to the initial AP at the root node of the search tree. However, when a large number of subproblems is expanded, the differences between the current AP and the AP at the top node grow. The solution time needed to solve these APs approach $O(n^3)$.

If we compare our results to the other algorithms in Fischetti *et al.* (2002), the FT-add method is slightly faster for many instances. This algorithm also applies relaxations different from the AP. As a consequence, it is able to keep search trees small for most instances, and keep the computational overhead within reasonable limits. Unfortunately, a direct comparison on the same computer cannot be made, since, to the best of our knowledge, no online source code is available. The Concorde solver and the FT Branch and Cut solver in Fischetti *et al.* (2002) both run under C-Plex. Both algorithms appear to be slower than our algorithms for randomly generated instances, but faster for the instances from the TSP library.

In a comparison among the DFS algorithms, Tables 3.7, 3.8, and 3.9 show that $BnB(SCS)$ obtains the smallest solution times for many instances, sparse instances, and instances with degree of symmetry 0.33 and 0.66, but the solution times of $BnB(Cost)$ are slightly better for fully symmetric instances. For sparse instances, the solution times of $BnB(SCS)$ are also shorter, but the advantage reduces as sparsity increases. $BnB(ECS)$, using the ECS lower bound and branching rule, generally requires too much tolerance calculation time to be competitive, in spite of its small search trees. In Table 3.10 and Table 3.11, $BnB(SCS)$ displays the fastest solution times, even though the instances with $k' = 5\%$ are almost symmetric.

Finally, we analyze the source of the search tree reductions of $BnB(ECS)$ and $BnB(SCS)$. A BnB algorithm first finds an optimal solution and then proves the optimality of that solution, i.e., all remaining subproblems are discarded. Table 3.12 shows that $BnB(Cost)$ spends a relatively large amount of time on finding an optimal solution compared to $BnB(ECS)$ and $BnB(SCS)$, particularly for non-symmetric random instances. This result indicates that the improved branching of tolerance-based algorithms is the predominant source of the search tree reductions. Table 3.12 also indicates that fast algorithms often spend the smallest fraction of time on finding an optimal solution. The value of an optimal solution is the tightest possible upper bound and can be used to fathom a large number of subproblems. We conclude that accurate branching is the key to good performance of Depth First Search BnB algorithms. Moreover, since tolerance-based algorithms find optimal solutions faster, the results in case of premature termination are likely to be better than for cost-based algorithms.

**Table 3.12.** Percentage of subproblems solved before an optimal solution is found

| Instance | $BnB(Cost)$ | $BnB(ECS)$ | $BnB(SCS)$ |
|---|---|---|---|
| ATSPLIB | 30.48% | 64.48% | 39.63% |
| Degree of symmetry 0.33 | 92.60% | 62.59% | 70.42% |
| Degree of symmetry 0.66 | 85.12% | 49.14% | 61.65% |
| Full symmetry | 35.70% | 31.71% | 32.08% |
| Asymmetric random | 90.02% | 77.27% | 75.76% |
| Degree of sparsity 50% | 95.15% | 82.49% | 76.88% |
| Degree of sparsity 75% | 96.56% | 80.77% | 77.65% |
| Degree of sparsity 90% | 95.83% | 72.30% | 82.35% |
| Buriol, $k' = 5$ | 82.57% | 62.25% | 66.16% |
| Buriol, $k' = 50$ | 97.27% | 52.47% | 41.93% |

This property may be very useful in case of solving large problems within limited times; see Zhang (1993).

There seems to exist the following paradox. The use of *lower bounds* based on tolerances cause the largest search tree reductions according to Goldengorin *et al.* (2004), and hence, one may expect that these algorithms need less time to prove the optimality of the solution at hand. However, Table 3.12 points out that tolerance-based BnB algorithms need relatively less time to find an optimal solution. An explanation is that the new lower bounds cut off a large number of subproblems *before* an optimal solution is found.

## 3.10 Summary and Future Research Directions

We presented an experimental analysis of tolerance-based BnB type algorithms for the ATSP, and compared it with cost-based BnB algorithms. Tolerance-based algorithms reduce the search tree sizes substantially. For random instances, including both instances with degree of symmetry 0.33 and 0.66, and sparse instances, the computation times are substantially better.

The better performance of tolerance-based BnB algorithms is mainly caused by improved *branching*: a better choice of entries to be included and excluded. Upper tolerances provide better predictions of which arcs in a relaxation solution should be preserved, the *survival set*, and which arcs should be deleted, the *extinction set*.

We apply the concept of *offenders*: sources of infeasibility which must be removed from a relaxation solution in order to obtain a feasible solution for the original hard problem. The minimal cost of removing such an offender can

be determined using tolerance values. This idea is used to construct new lower bounds, but it also enables the BnB algorithm to make branching steps with large increases in solution value without "jumping" across an optimal ATSP solution. The largest increase in lower bound is obtained if we consider all offenders, the *Entire Cycle Set*, which has of course the drawback of very long tolerance calculation times. It is shown that a good approximation is the *Smallest Cycle Set*, which only uses a smallest cycle in the set of offenders, so that only a few tolerances need to be calculated. Branching on the smallest cycle is a good choice, not only because a small number of new subproblems is generated, but also because it is very likely that a large branching step towards an optimal ATSP solution is made.

Tolerance-based BnB algorithms have one major drawback: they have to compute tolerances at every node of the search tree. In spite of this drawback, it turns out that the BnB algorithm with the Smallest Cycle Set bound and branching rule usually require shorter solution times than cost-based BnB algorithms. This algorithm is faster for difficult instances than the state-of-the-art BnB algorithm from Carpaneto *et al.* (1995).

The idea of branching on tolerances can be seen as similar to the idea of *strong branching* in Integer Programming; see Achterberg *et al.* (2004). Strong branching first explores the additional cost of setting a fractional variable at an integer value, and then decides on which variable to branch. Similar to 0tolerance-based branching, it first computes the additional cost of removing infeasibilities, the fractional value of a variable, before it takes the branching step. In Achterberg *et al.* (2004), it is found that strong branching should be done only at specific nodes of the search tree. Similar strategies can be developed for tolerance-based algorithms.

An interesting direction of research is to develop book-keeping techniques that accelerate tolerances computations, and lead to solution time reductions for ATSP instances. Another direction of research is to construct tolerance-based algorithms for other COPs. Preliminary experiments with these algorithms are very promising as well.