UNIVERSITÀ DEGLI STUDI DI BOLOGNA

DEIS - Dipartimento di Elettronica Informatica e Sistemistica

Dottorato di Ricerca in
Automatica e Ricerca Operativa
(MAT/09)

XVIII Ciclo

LP-based Heuristics for the Traveling Salesman Problem

Tesi di Dottorato di:

Matteo Fortini

Il Coordinatore	**Il Supervisore**
Ch.mo Prof. Claudio Melchiorri	Ch.mo Prof. Andrea Lodi Ch.mo Prof. Paolo Toth

Bologna, 15 Marzo 2007

A mia moglie Elena
e alle nostre bimbe Alice e Irene

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The *Symmetric Traveling Salesman Problem*, or STSP, is the problem of finding a minimum weight Hamiltonian circuit in an edge-weighted graph. The STSP is a fundamental problem in combinatorial optimisation, and has received a huge amount of attention in the literature (see the books Lawler et al. [15] and Gutin & Punnen [13] and the survey papers by Jünger, Reinelt & Rinaldi [84, 85]).

The STSP is strongly $\mathcal{NP}$-hard, which means that heuristics must be used to tackle hard instances. Yet, it is possible to solve suprisingly large instances to optimality via the so-called *branch-and-cut* technique; see for example Padberg & Rinaldi [121] or Applegate et al. [18]. The first step in the branch-and-cut approach is normally to solve the so-called *subtour relaxation* (or Held-Karp relaxation), giving a lower bound to the cost of the optimal tour. In practice, this lower bound is fairly strong, and it has been conjectured (see for example Goemans [71]) that the worst-case ratio between the optimum and the lower bound is never more than $4/3$.

It seems reasonable to suppose that the solution to the subtour relaxation, though typically fractional, contains some information which could be exploited by a heuristic method. This is the idea underlying the *compatible tour* heuristic, which is the topic of this thesis.

The structure of the remainder of the thesis is as follows. The heuristic is formally defined in Section 4.2. In Chapter 5, we prove two negative results: that the problem of finding the best compatible tour is strongly $\mathcal{NP}$-complete, and that the worst-case ratio between the cost of the best compatible tour and that of the optimum can approach $5/3$, even when the instance is metric (i.e., has edge costs satisfying the triangle inequality). In Chapter 6, we present some structural properties which are satisfied by solutions to the subtour relaxation, and show how these can be used to devise algorithms for finding the best compatible tour. Finally, computational results are given in Chapter 7.

Chapter 2

Traveling Salesman Problem (TSP)

The Traveling Salesman Problem can be worded as:

> Given a finite set of cities and the distances between each pair of them, find the shortest tour that passes through each city only once and goes back to the first one in the end.

A more mathematical formulation is

> Find the shortest hamiltonian tour in a finite, complete, weighted graph.

2.1 Some history

Even if the traveling salesman problem can be found in several practical fields, it is still unclear how and when it started to interest mathematicians from a theoretical point of view.

2.1.1 A handbook from 1832

One of the first explicit formulation of a "Traveling salesman problem" is found in a handbook from 1832, dedicated to traveling salespersons, titled "The traveling salesman – how he should be and what he should do, to get the orders and assure success for his business – from and old traveling salesman".

The problem is described in this way:

> Business brings the traveling salesman first here, then there, and it is not possible to describe paths that adapt to every possible case; sometimes however, thanks to a careful choice of paths and to an appropriate sequence of travels, one can save so much time, that we can't avoid showing some rules on this subject.
>
> Anyone can take from these rules as much as he deems useful to his objectives; we believe, however, that we can largely insure, that it may not be possible to order the paths through Germany in a more economical

> way in distance terms and, what must be of primary importance to the traveler, regarding the length of the travel forth and back. The main issue is visiting the biggest number of places, without having to pass through anyone of them more than once.

2.1.2 Karl Menger, 1920-1930

The first mathematician who wrote anything about the TSP seems to be Karl Menger, who solved related problems during his studies about the length of a curved line in a metric space.

In a speech he gave in Vienna in 1930, Menger talks about what he calls the *Botenproblem*:

> We call the *messenger problem (Botenproblem)* (since in practice this problem should be solved by any postman, and anyhow also by many travelers) the task of finding, for a finite number of points for which the relative distances are known, the shortest path that connects them. Surely, the problem can be solved using an enormous number of tentatives. There are no known rules that allow to lower the number of tentatives below the number of permutations of the given points. The rule of going from the starting point to the nearest, then to the nearest to the reached point, etc., doesn't lead to the shortest tour.

Note that Menger claimed that a *greedy* algorithm is not optimal for the TSP.

2.1.3 Harvard, Princeton 1930-1934

The history of the TSP in these years is not clear, primarily regarding the events' dates.

What is sure is that Menger was visiting lecturer at the Harvard University from September, 1930 to February, 1931, and in one of his seminars he showed his results on the length of curved lines, including his formulation of the *messenger problem*.

Hassler Whitney, then researcher in graph theory at the Harvard University, took on the problem, suggesting its practical implications.

It was Whitney who brought the problem to Princeton University, where he set the problem of finding the shortest tour among the 48 American States.

Also in Princeton, Flood and Tucker started to note the similarities between the TSP and the hamiltonian tour problem, which can be formulated as:

> Given a connected graph, find a tour that passes through each node only once.

The TSP is a natural generalization of the hamiltonian tour problem, since it asks for the shortest hamiltonian tour.

2.1.4 The year 1940

The first mathematical papers about the TSP go back to 1940. The main problem the research focus on is finding a lower-bound for the best tour. The reason behind the interest came from an expedition in Bengal, for which one of the most expensive factor was the movement of men and material from a place to some other.

After that, mathematicians generalized the problem, first to a finite number of random points in a unity-sided square, then in a general area.

In this period Ghosh noted that the problem proved to be very complex, except for very small instances, that were although scarcely interesting in practice.

2.1.5 RAND Corporation 1947-1952

Merrill Flood brought the traveling salesman problem to RAND Corporation in Santa Monica, CA.

People in RAND tried to apply known methods for transportation and assignment problems to the TSP, without success. RAND Corporation promised a reward to anyone that found a fundamental theorem about the TSP.

2.1.6 Dantzig, Fulkerson, Johnson 1954

RAND researchers RAND Dantzig, Fulkerson and Johnson were the first to set a milestone in the solution of the TSP.

Applying the simplex method recently developed by Dantzig, they solved a 49 city problem in the USA very similar to the one stated by Whitney, with a city in every state, plus Washington, D.C.

The cutting plane method for solving LP problems was developed in this period.

2.1.7 1954 – today

After the solution of the 49 city problem by Dantzig, Fulkerson and Johnson, the improvements on the solution of the TSP have come mainly from the development of improved methods of applying the cutting plane method to the LP. However, there have been no sensational theoretical results, so that the reward posed by the RAND Corporation is still to be claimed.

The theoretical work has been focused on a better knowledge about the TSP polytope and its properties, in order to find better cutting planes. On the other side there has been the development of better and better heuristic algorithm to find quasi-optimal solutions.

One of the most important theoretical results is the proof by Richard Karp, Eugene Lawler and Robert Tarjan of the $\mathcal{NP}$-hardness of the decision version of the TSP in 1972. [1]

[1] We briefly recall that the proof of th $\mathcal{NP}$-hardness of a problem $\mathcal{P}$ is done in two steps:

1. $\mathcal{P}$ can be solved in polynomial time *in the best case*

Today, the research on the TSP is very active and its results have been successfully applied to other Operations Research problems.

On one side, there is the search for better heuristics, both for speed and for optimality, using for example simulated annealing, genetic algorithms, tabu-search, neural networks or ant-colonies.

On the other side, researchers have developed better and better software to solve the TSP to optimality, mainly through variants of the cutting plane method. CONCORDE, which we used for our research, is the best software developed for the TSP in term of the size of solved problems, and contains a large part of the recent algorithmic results on the TSP.

The rating to a software designed for solving the TSP is given by its ability of solving previously unsolved instances, or by its bigger speed in solving already solved problems compared to existing softwares.

A common test body of instances is being collected by Gerhard Reinelt in a library called TSPLIB.

We cite an interesting observation by the authors of CONCORDE about the research on the TSP:

> Arguing that [the TSP] is popular because it arises from practical applications would be hard: even though variations on the TSP theme come up in practice relatively often, the theme in its pure form appears only rarely. Two of its sources are (a) drilling holes in printed circuit boards, where the time spent on moving the drill through a sequence of prescribed positions is to be minimized, and (b) X-ray crystallography, where the time spent on moving the diffractometer through a sequence of prescribed angles is to be minimized. [...]
>
> Writing computer programs to solve TSPLIB problems can hardly be classified as applied work. The technology of manufacturing printed circuit boards has changed and the drilling problems from the TSPLIB are no longer of interest to the industry. Similarly, development of multiplex cameras is making the TSP problems of X-ray crystallography obsolete; besides, research laboratories have never considered large investments of computer time for the small gain of minimizing the time spent by moving the diffractometer. Furthermore, even if there were a client with a genuine need to solve TSP problems, such a client would most likely be satisfied with nearly optimal tours. Finding nearly optimal tours even in fairly large TSPLIB problems is a relatively easy task: good implementations of the Lin-Kernighan heuristic and its refinements work like a charm. Most of the computer time spent on solving TSPLIB problems goes into proving that a tour is optimal, a fact of negligible interest to the hypothetical client.

2. $\exists \mathcal{P}' : \mathcal{P}' \in \{\mathcal{NP}\text{-hard problems}\} \wedge \mathcal{P}' \propto \mathcal{P}$
$\Leftrightarrow (\forall \mathcal{P}'\ \mathcal{P}' \in \{\mathcal{NP}\text{-hard problems}\} \Rightarrow \mathcal{P}' \propto \mathcal{P})$

Writing computer programs to solve TSPLIB problems can hardly be classified as theoretical work, either. A prize offered by the RAND corporation for a significant theorem bearing on the TSP was never awarded; Dantzig, Fulkerson, and Johnson close their seminal paper with the modest disclaimer: "It is clear that we have left unanswered practically any question one might pose of a theoretical nature concerning the traveling-salesman problem". All the successful computer programs for solving TSP problems follow the Dantzig-Fulkerson-Johnson scheme; improvements consist only of better ways of finding cuts and better handling of the large linear programming relaxations. (Having a faster computer helps, too.)

Writing computer programs to solve TSPLIB problems could be classified as a sport , where each new record is established by solving at least one previously unsolved instance. [...] [17]

Chapter 3

Linear Programming formulation of the TSP

The simplex method and the increasing performances of computers make easy to solve quite large continuous LP instances to the optimum.

A natural way of solving the TSP to optimality has thus been writing it as a mixed integer LP problem.

3.1 The TSP and Mixed-Integer Programming in general

3.1.1 Introduction

Many problems in science, technology, business, and society can be modeled as mixed integer programming (MIP) problems and, as a matter of fact, in the last decade the use of integer programming models and software has increased dramatically. Nowadays, thanks to the progress of computer hardware and, even more, advances in the solution techniques and algorithms, it is possible to solve problems with thousands of integer variables on personal computers, and to obtain high quality solutions to problems with millions of variables (for example, set partitioning problems) often in a matter of minutes.

Among the currently most successful methods, to solve MIP problems, are linear programming (LP, for short) based branch-and-bound algorithms, where the underlying linear programs are possibly strengthened by cutting planes.

Today's codes, however, have become increasingly complex with the incorporation of sophisticated algorithmic components, such as preprocessing and probing techniques, cutting plane algorithms, advanced search strategies, and primal heuristics.

3.1.2 Definitions

A *mixed integer program* (MIP) is a system of the following form:

$$\begin{array}{rlll} z_{\mathsf{MIP}} = & \min & c^T x & \\ & \text{subject to} & Ax \leq b & \\ & & \underline{x} \leq x \leq \overline{x} & \\ & & x \in \mathbb{Z}^{\mathcal{G}} \times \mathbb{R}^{\mathcal{C}}, & \end{array} \tag{3.1}$$

where $A \in \mathbb{Q}^{\mathcal{M}\times(\mathcal{G}\cup\mathcal{C})}$, $c \in \mathbb{Q}^{\mathcal{G}\cup\mathcal{C}}$, $b \in \mathbb{Q}^{\mathcal{M}}$. Here, $c^T x$ is the *objective function*, $Ax \leq b$ are the *constraints* of the MIP, and $\mathcal{M}$, $\mathcal{G}$ and $\mathcal{C}$ are non-empty, finite sets with $\mathcal{G}$ and $\mathcal{C}$ disjoint. Without loss of generality, we may assume that the elements of $\mathcal{M}$, $\mathcal{G}$ and $\mathcal{C}$ are represented by numbers, i.e., $\mathcal{M} = \{1, \ldots, m\}$, $\mathcal{G} = \{1, \ldots, p\}$ and $\mathcal{C} = \{p+1, \ldots, n\}$. The vectors $\underline{x} \in (\mathbb{Q} \cup \{-\infty\})^{\mathcal{G}\cup\mathcal{C}}, and \overline{x} \in (\mathbb{Q} \cup \{\infty\})^{\mathcal{G}\cup\mathcal{C}}$ are called *lower* and *upper bounds* on x, respectively. A variable $x_j, j \in \mathcal{G} \cup \mathcal{C}$, is *unbounded from below* (*above*), if $\underline{x}_j = -\infty$ ($\overline{x}_j = \infty$). An integer variable $x_j \in \mathbb{Z}$ with $\underline{x}_j = 0$ and $\overline{x}_j = 1$ is called *binary*. If $\mathcal{G} = \emptyset$ then (3.1) is called *linear program* or *LP*. If $\mathcal{C} = \emptyset$ then (3.1) is called *integer program* or *IP*. A 0-1 MIP is a MIP where all the integer variables are binary. A vector x that satisfies all constraints is called a *feasible solution*. An *optimal solution* is a feasible solution for which the objective function achieves the smallest value.

From a complexity point of view mixed integer programming problems belong to the class of NP-hard problems (see Garey and Johnson [131], for example) which makes it unlikely that efficient, i.e., polynomial time, algorithms for their solution exist.

The *linear programming (LP) relaxation* of a MIP is the problem obtained from (3.1) by dropping the integrality restrictions, i.e., replacing $x \in \mathbb{Z}^{\mathcal{G}} \times \mathbb{R}^{\mathcal{C}}$ with $x \in \mathbb{R}^{\mathcal{G}\cup\mathcal{C}}$. The optimal value of the LP relaxation provides a lower bound on the optimal value of the MIP. Therefore, if an optimal solution to the LP relaxation satisfies the integrality restrictions, then that solution is also optimal for the MIP. If the LP relaxation is infeasible, then the MIP is also infeasible, and if the LP relaxation is unbounded, then the MIP is either unbounded or infeasible. If the MIP is feasible and its LP relaxation is bounded, then the MIP has optimal solution(s)[1].

A common approach to solving a MIP consists of solving its LP relaxation[2] in the hope of finding an optimal solution x^* which happens to be integer. If this is not the case, there are two main ways to proceed. In the *cutting-plane* approach, one enters the *separation phase*, where a linear inequality (*cut*) $\alpha^T x \leq \alpha_0$ is identified which separates x^* from the feasible solutions of the MIP. The cut is appended to the current LP relaxation, and the procedure is iterated. In the *branch-and-bound* approach, instead, the MIP is replaced by two subproblems obtained, e.g., by imposing an additional restriction of the type $x_j \leq \lfloor x_j^* \rfloor$ and $x_j \geq \lceil x_j^* \rceil$, respectively, where x_j

[1] This is in general not true when the problem is not rational. Consider for example the problem $\max\{x_1 - \sqrt{2}x_2 : x_1 \leq \sqrt{2}x_2, x_1, x_2 \text{ integer}\}$; this has feasible solutions with negative objective values arbitrarily close to 0, but none equal to 0.

[2] Linear programs can efficiently be solved using Dantzig's simplex algorithm or interior point methods. For an introduction to linear programming see, for instance, Nemhauser and Wolsey [111], or Bertsimas and Tsitsiklis [132].

is an integer-constrained variable with fractional value in x^*. The procedure is then recursively applied to each of the subproblems.

3.1.3 Preprocessing

Preprocessing is the name for a number of techniques, employed by MIP solvers, aimed at reducing the size of an instance and strengthen its LP bound [133]. Preprocessing can alter a given formulation quite significantly by fixing, aggregating, and/or substituting variables and constraints of the problem, as well as changing the coefficients of the constraints and the objective function.

As an example (taken from Martin [134]), consider the following *bounds strengthening* technique, where we exploit the bounds on the variables to detect so-called forcing and dominated rows. Given some row i, let

$$\begin{aligned} L_i = &\sum_{j \in P_i} a_{ij}\underline{x}_j + \sum_{j \in N_i} a_{ij}\overline{x}_j, \\ U_i = &\sum_{j \in P_i} a_{ij}\overline{x}_j + \sum_{j \in N_i} a_{ij}\underline{x}_j \end{aligned} \tag{3.2}$$

where $P_i = \{j : a_{ij} > 0\}$ and $N_i = \{j : a_{ij} < 0\}$. Obviously, $L_i \leq \sum_{j=1}^{n} a_{ij}x_j \leq U_i$. The following cases might come up. An inequality i is an *infeasible row* if $L_i > b_i$. In this case the entire problem is infeasible. An inequality i is a *forcing row* if $L_i = b_i$. In this case all variables in P_i can be fixed to their lower bound and all variables in N_i to their upper bound. Row i can be deleted afterwards. An inequality i is a *redundant row* if $U_i < b_i$. In this case i can be removed.

This row bound analysis can also be used to strengthen the lower and upper bounds of the variables. Compute for each variable x_j and each inequality i

$$\begin{aligned} u_{ij} = &\begin{cases} (b_i - L_i)/a_{ij} + \underline{x}_j, & \text{if } a_{ij} > 0 \\ (L_i - U_i)/a_{ij} + \underline{x}_j, & \text{if } a_{ij} < 0 \end{cases} \\ l_{ij} = &\begin{cases} (L_i - U_i)/a_{ij} + \overline{x}_j, & \text{if } a_{ij} > 0 \\ (b_i - L_i)/a_{ij} + \overline{x}_j, & \text{if } a_{ij} < 0. \end{cases} \end{aligned} \tag{3.3}$$

Let $u_j = \min_i u_{ij}$ and $l_j = \max_i l_{ij}$. If $u_j \leq \overline{x}_j$ and $l_j \geq \underline{x}_j$ we speak of an *implied free variable*. The simplex method might benefit from not updating the bounds but treating variable x_j as a free variable (note that setting the bounds of x_j to $-\infty$ and $+\infty$ will not change the feasible region). Free variables will commonly be in the basis and are thus useful in finding a starting basis. For mixed integer programs however, if the variable x_j is integer, it is better in general to update the bounds by setting $\overline{x}_j = \min\{\overline{x}_j, u_j\}$ and $\underline{x}_j = \max\{\underline{x}_j, l_j\}$, because the search region of the variable within an enumeration scheme is reduced. In case x_j is an integer (or binary) variable we round u_j down to the next integer and l_j up to the next integer. For example consider the following inequality:

$$-45x_6 - 45x_{30} - 79x_{54} - 53x_{78} - 53x_{102} - 670x_{126} \leq -443 \tag{3.4}$$

Since all variables are binary we get $L_i = -945$ and $U_i = 0$. For $j = 126$ we obtain $l_{ij} = (-443 + 945)/ - 670 + 1 = 0.26$. After rounding up it follows that x_{126} must be 1. Note that with these new lower and upper bounds on the variables it might pay to recompute the row bounds L_i and U_i, which again might result in tighter bounds on the variables.

Techniques that are based on checking infeasibility are called primal reduction techniques. Dual reduction techniques make use of the objective function and attempt to fix variables to values that they will take in any optimal solution.

One preprocessing technique that may have a big impact in strengthening the LP relaxation of a MIP formulation is *coefficient improvement*. This technique updates the coefficients of the formulation so that the constraints define a smaller LP relaxation, hence leading to improved LP bounds (see Nemhauser and Wolsey [111], for example).

Probing is another technique that considers the implications of fixing a variable to one of its bounds. For instance, if fixing a binary variable x_1 to one, forces a reduction in the upper bound of x_2, then one can use this information in all constraints in which x_2 appears and possibly detect further redundancies, bound reductions, and coefficient improvements.

See, for example, Savelsbergh [133] or Martin [134] for a survey of these and other preprocessing techniques.

Finally, all these techniques can be applied not only before solving the initial formulation at the root node, but also before each sub-problem in the branching tree. However, since the impact of node preprocessing is limited to only the subtree defined by the node, one should take into consideration whether the time spent on node preprocessing is worthwhile.

3.1.4 Valid Inequalities and Cutting Planes

A *valid inequality* for a MIP is an inequality that is satisfied by all feasible solutions. A *cutting plane*, or simply *cut*, is a valid inequality that is not satisfied by all feasible points of the LP relaxation. Thus, if we find a cut, we can add it to the formulation and strengthen the LP relaxation.

In the late 50's, Gomory [135] pioneered the cutting-plane approach, proposing a very elegant and simple way to derive cuts for an IP by using information associated with an optimal LP basis. This was later generalized by Chvátal [46] (see also Wolsey [136], for instance).

We can construct a valid inequality for the set $X := P \cap \mathbb{Z}^n$, where $P := \{x \in \mathbb{R}^n_+ : Ax \leq b\}$, and A is an $m \times n$ matrix, as follows. Let u be an arbitrary vector in $\mathbb{R}^m_+$. Then the inequality

$$u^T Ax \leq u^T b \tag{3.5}$$

is a valid inequality for P. Since $x \geq 0$ in P, we can round down the coefficients of the left-hand side of (3.5) and obtain that the inequality

$$\lfloor u^T A \rfloor x \leq u^T b. \tag{3.6}$$

Finally, as x is integer in X, we can also round down the right-hand side and get

$$\lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor \tag{3.7}$$

as a valid inequality for X.

Quite surprisingly, all valid inequalities for an integer program can be generated by applying repeatedly this procedure a finite number of times (for a proof see Chvátal[46], Schrijver [138], or Wolsey [136]).

In principle MIPs can be solved to optimality using the cutting plane algorithm of Algorithm 1, however practical experience with Gomory's algorithm shows that the quality of the cuts generated becomes rather poor after a few iterations, which causes the so called *tailing-off* phenomenon: a long sequence of iterations without significant improvements towards integrality. Adding too many cutting planes also leads to numerical problems, thus a stopping criterion is used to terminate the generation of cuts, even if a feasible solution is not reached.

```
1 repeat
2     solve the current LP relaxation
3     if the optimal solution x* is integer feasible then
4         return x*
5     else
6         find a cutting plane (π, π_0) violated by x*
7         add the cut π^T x ≤ π_0 to the current formulation
8     end
9 until stopping criterion reached ;
```

Algorithm 1: Generic cutting plane algorithm

The first successful application of a cutting plane algorithm is due to Dantzig, Fulkerson, and Johnson [56] who used it to solve large (for that time) instances of the traveling salesman problem.

The cutting planes implemented in MIP solvers can be classified into two broad categories. The first are general cuts that are valid for any MIP problem; these include Gomory mixed-integer and mixed-integer rounding cuts [137, 139]. The second category includes strong polyhedral cuts from knapsack [140, 141, 142, 154, 155], fixed-charge flow [156, 143] and path [144], and vertex packing [145] relaxations of MIPs. Strong inequalities for these simpler substructures are usually quite effective in improving LP relaxations of more complicated sets. MIP solvers automatically identify such substructures by analyzing the constraints of the formulation and try to add appropriate cuts.

3.1.5 Branch-and-Bound

Branch-and-bound algorithms for mixed integer programming use a "divide and conquer" strategy to explore the set of all feasible mixed integer solutions. These algorithms build a search tree, in which the nodes of the tree represent subproblems defined over subsets of the feasible region. According to Wolsey [136] the first paper presenting a branch-and-bound strategy for the solution of integer programs is due to Land and Doig [146].

Let P_0 be a mixed-integer programming problem of the form (3.1). Let $X_0 := \{x \in \mathbb{Z}^p \times \mathbb{Q}^{n-p} : Ax \leq b, \underline{x} \leq x \leq \overline{x}\}$ be the set of feasible mixed integer solutions of problem P_0. If it is too difficult to compute

$$\begin{array}{rll} z_{\text{MIP}} = & \min & c^T x \\ & \text{subject to} & x \in X_0, \end{array} \tag{3.8}$$

then we can split X_0 into a finite number of disjoint subsets $X_1, \ldots, X_k \subset X$, such that $\cup_{j=1}^{k} X_j = X_0$, and try to solve separately each of the subproblems

$$\begin{array}{ll} \min & c^T x \\ \text{subject to} & x \in X_j, \quad \forall\, j = 1, \ldots, k. \end{array} \tag{3.9}$$

Afterwards we compare the optimal solutions of the subproblems and choose the best one. Since each subproblem is usually only slightly easier than the original problem, this idea is iterated recursively splitting the subproblems again into further subproblems. The (fast-growing) list of all subproblems is usually organized as a tree, called a *branch-and-bound tree* and we say that a *father* or *parent* problem is split into two or more *son* or *child* problems. This is the branching part of the branch-and-bound method.

For the bounding part of this method we assume that we can efficiently compute a lower bound $z^*(P)$ of each subproblem P (with feasibility set X), so that $z^*(P) \leq \min_{x \in X} c^T x$. In the case of mixed integer programming this lower bound can be obtained by using the *LP relaxation*.

During the exploration of the search tree we can find that the optimal solution x^* of the LP relaxation of a subproblem P is also a feasible mixed integer point, i.e., $x^* \in X$. When x^* is not feasible, it is sometimes possible to obtain a feasible point by rounding the integer variables, or using more advanced heuristics. The feasible solution with the smallest objective value, z^{best}, found so far is called the *incumbent solution*. This allows us to maintain an upper bound on the optimal solution value z_{MIP} of P_0, as $z_{\text{MIP}} \leq z^{best}$. Having a good upper bound is crucial in a branch-and-bound algorithm, because it keeps the branching tree small. In fact, suppose the solution of the LP relaxation of some other subproblem P' satisfies $z^*(P') \geq z^{best}$, then the subproblem P' can be pruned, without further processing, because the optimal solution of this subproblem cannot be better than the incumbent one.

Algorithm 2 summarizes the whole branch-and-bound procedure for mixed-integer programs.

```
1  let L := {P_0}
2  let z^best := +∞
3  repeat
4      select a problem P from L
5      remove P from L
6      solve the LP relaxation of P
7      if LP feasible then
8          let x* be an optimal solution of P
9          let z*(P) := c^T x*
10         if x* feasible (for P_0) then
11             if z^best > z^opt(P) then
12                 let z^best := z*(P)
13                 let x̃ := x*
14                 delete from L all subproblems P with z*(P) ≥ z^best
15             end
16         else
17             split problem P into subproblems and add them to L
18         end
19     end
20 until L = ∅ ;
21 return z^best and x̃
```

Algorithm 2: Generic branch-and-bound algorithm

Initially the active node list L contains only the root node. Then, whenever a node is *processed*, its LP relaxation is solved and the node is either pruned or split into sub-problems which are added back to the list. Therefore, at any given time, the nodes in the list L correspond to the unsolved problems in the branch-and-bound tree, which are the leaves of the tree.

Within this general framework, there are two aspects that requires a choice to be taken. The first one is how to perform the *branching* at step 17, that is how split a problem P into subproblem. And the second one is how to choose which problem to process next at step 4 (*node selection*).

Branching

A natural way to divide the feasible region of a MIP problem is to choose a variable x_i that is fractional in the current linear programming solution x^* and create two subproblems, one with the updated bound $x_i \leq \lfloor x_i^* \rfloor$ and the other with $x_i \geq \lceil x_i^* \rceil$. This type of branching is referred to as *variable dichotomy*.

In general there are several fractional variables to choose from. Since the effectiveness of the branch-and-bound algorithm depends heavily on the convergence of upper and lower bounds, we would like to choose the variable that leads to the highest bound improvement. However this has been proved to be "difficult" so what is typically done is select a list of "candidate" branching variables, among those that are most fractional, and then, for each of these, estimate the LP bound that a branching on that variable would lead to. One of the methods used to estimate the bound improvement consists in performing a small number of pivots and observe what happens to the objective function (*strong branching*).

Node selection

There are two main possible strategies to visit the branching tree. In the *best-first* search, the node P with the lowest $z^*(P)$ is chosen, since it is supposed to be closer to the optimal solution. At the other extreme is the *depth-first* search: nodes are ordered according to their depth in the branching tree, and the deepest pending node is processed first.

For a fixed branching rule, best-first search minimizes the number of nodes evaluated before completing the search. However, there are two main drawbacks: one is that the search tends to stay in the higher levels of the branching tree, where problems are less constrained and, thus, hardly lead to improvements of the incumbent solution. The second one is that the search tree tends to be explored in a breadth-first fashion, so subsequent linear programs have little relation to each other, leading to longer evaluation times. One way to mitigate this second issue could be to save the basis information at all the nodes, but in this case the memory requirements for searching the tree in a best-first manner might become prohibitive.

Depth-first is easier to implement, has lower memory requirement, and changes in the linear program, from one node to the next, are minimal (only a variable bound). It also usually finds feasible solutions more quickly than with best-first search as feasible

solutions are typically found deep in the search tree. One disadvantage is that, when a "bad" branch (i.e., not containing good feasible solutions) is visited, this strategy searches exhaustively the whole sub-tree before backtracking to different areas. Also, it can spend a lot of time solving nodes that could have been pruned if a better incumbent had been known.

Most integer programming solvers employ a hybrid of best-first search and depth-first search, trying to benefit from the strengths of both, regularly switching between the two strategies during the search. In the beginning the emphasis is usually more on depth-first, to find high quality solutions quickly, whereas in the later stages of the search, the emphasis is usually more on best-first, to improve the lower bounds.

3.1.6 Branch-and-Cut

Combination of cutting-plane and branch-and-bound techniques was attempted since the early 70's. Initially, however, the constraint generators were used only at the root node, as a simple preprocessor, to obtain a tighter LP relaxation of the original MIP formulation. In the mid 80's, Padberg and Rinaldi [118, 121] introduced a new methodology for an effective integration of the two techniques, which they named *branch-and-cut*. This is an overall solution scheme whose main ingredients include: the generation at every node of the branching tree of (facet-defining) cuts globally valid along the tree; efficient cut management by means of a constraint pool structure; column/row insertion and deletion from the current LP; variable fixing and setting; and the treatment of inconsistent LP's.

Branch-and-cut has a number of advantages over pure cutting-plane and branch-and-bound schemes. With respect to the branch-and-bound approach, the addition of new cuts improves the LP relaxation at every branching node. With respect to the pure cutting-plane technique, one can resort to branching as soon as tailing-off is detected. As the overall convergence is ensured by branching, the cut separation can be of heuristic type, and/or can restrict to subfamilies of problem-specific cuts which capture some structures of the problem in hand. Moreover, the run-time variable pricing and cut generation/storing mechanisms allow one to deal effectively with tight LP relaxations having in principle a huge number of variables and constraints.

While a branch-and-cut algorithm spends more time in solving the LP relaxations, the resulting improved LP bounds usually leads to a significantly smaller search tree. Naturally, as with all techniques designed to improve the performance of the basic branch-and-bound algorithm, the time spent on cut generation must be contained in order not to outweigh the speed-up due to improved LP bounds.

For more information about branch-and-bound and branch-and-cut see also Nemhauser and Wolsey [111], for instance.

3.1.7 Primal Heuristics

In order to reduce the size of the branching tree, it is very useful to find good incumbent solutions as soon as possible. On the other hand, waiting to find feasible

solutions at a node just by solving its LP relaxations can take a very long time. Therefore MIP solvers attempt to find feasible solutions early in the search tree by means of simple and quick heuristics. As an extreme example, if the optimal solution would be known at the root node, then branch-and-bound would be used only to prove the optimality of the solution. In this case, for a fixed branching rule, any node selection rule would produce the same tree with the minimum number of nodes.

While cutting planes (and, to some extent, bounds) are used to strengthen the lower bound on the optimal solution, primal heuristics have the complementary role of improving the upper bound, given by incumbent solutions, and help close the gap between the two.

Finding a feasible solution of a given MIP model is, however, a very important (NP-complete) problem that can be extremely hard in practice. Several techniques are typically used, involving simple rounding, partial enumeration, diving (into the branching-tree) and variable fixing. Once one feasible solution has been found, other *improvement algorithms* can be used to iteratively try to obtain a better solution. Neighborhood search algorithms (alternatively called local search algorithms) are a wide class of improvement algorithms where at each iteration an improving solution is found by searching the "neighborhood" of the current solution.

Since these procedures can be quite time consuming, it seems reasonable to spend more effort on finding good feasible solutions early in the search tree, since this would have the most impact on the solution process. Very recently, Fischetti, Glover and Lodi proposed a heuristic scheme for finding a feasible solution to 0-1 MIPs, called *Feasibility Pump* (FP).

3.1.8 Truncated search

While mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems of practical interest, the exact solution of the resulting models often cannot be carried out (in a reasonable time) for the problem sizes of interest in real-world applications, hence one is interested in effective heuristic methods.

Although several heuristics have been proposed in the literature for specific classes of problems, only a few papers deal with general-purpose MIP heuristics, including [147, 148, 149, 150, 151, 152, 153, 157, 158] among others.

Exact MIP solvers are nowadays very sophisticated tools designed to hopefully deliver, within acceptable computing time, a provable optimal solution of the input MIP model, or at least a heuristic solution with a practically-acceptable error. In fact, what matters in many practical cases is the possibility of finding reasonable solutions as early as possible during the computation.

In this respect, the "heuristic behavior" of the MIP solver plays a very important role: an aggressive solution strategy that improves the incumbent solution at very early stages of the computation is strongly preferred to a strategy designed for finding good solutions only at the late steps of the computation (that, for difficult problems, will unlikely be reached within the time limit).

Many commercial MIP solvers allow the user to have a certain control on their heuristic behavior through a set of parameters affecting the visit of the branching tree, the frequency of application of the internal heuristics, the fact of emphasizing the solution integrality rather than its optimality, etc. Some recently proposed techniques (Local Branching [159] and RINS [160]), have provided a considerable improvement in this direction enhancing the heuristic behavior of MIP solvers through appropriate diversification mechanisms borrowed from local search paradigms.

Therefore, even if branch-and-cut algorithms are NP-hard, it may be reasonable to model a problem as a MIP instance and search for "heuristic" solutions by means of a black-box general purpose MIP solver with a truncated search - thus exploiting the level of sophistication reached nowadays by these tools.

3.1.9 Column generation

Another problem with MIP problems is the cost of storing an processing all the variables involved in the problem. For each variable the solver must store the corresponding (sparse) column vector in the *tableau* and must process it for every new cut inserted in the LP. For this reason, it's useful to keep the minimum needed number of variables in the LP, inserting variables that can lead to a better solution every time one finds an extreme point. The initial set of variables can be defined in several ways, but it's better if it contains all the variables involved in a feasible solution. In this way the LP won't become infeasible for cuts that are globally valid, reducing the time needed to search for variables to add. With column generation a point with no violated cuts in the current set of variables, could have some variables with possibly negative reduced cost, which means that they could lower the objective function if they entered the basis. Such variables can be found by computing the reduced costs of all the variables which are not in the LP using the dual values of the cuts in the LP. The reduced cost of a variable can be computed if the LP is feasible by

$$\bar{c}_{i,j} = c_{i,j} - \pi_i a_{i,j} \text{ for } i = 1, 2, \ldots, m$$

where π_i is the dual value of cut i. If the LP is infeasible, the variables that could make it feasible are still the ones with the most negative reduced costs, calculated as:

$$\bar{c}_{i,j} = -\pi_i a_{i,j} \text{ for } i = 1, 2, \ldots, m$$

The column generation phase looks for variables with negative reduced costs, then adds at least some of the most negative ones to try to lower the objective function. The reoptimization after having added the variables can lead to a better extremum point or can expose some violated cuts.

3.2 IP formulation of the TSP

A problem with n vertices must give $n\left(n-1\right)/2$ distances for each pair of vertices. Enumerating all the possible pairs, or *edges*, it is possible to write any instance of the

TSP as a *costs* vector c of size $n\,(n-1)\,/2$, containing the distance between each pair of vertices. Every feasible tour for the TSP can be written as an incidence vector x of size $n\,(n-1)\,/2$, with a variable for each pair of vertices, where $x_{i,j}$ is 1 if the edge $e_{i,j}$ is in the tour and 0 otherwise.

Calling $\mathcal{S}$ the set of all feasible solutions, the TSP becomes:

$$\text{min} \quad c^T x \quad \text{s.t.} \quad x \in \mathcal{S} \tag{3.10}$$

Problem (3.10) is a MIP problem, so it is intrinsecally NP-hard.

Dantzig and his colleagues changed the stated problem to a linear programming one, with constraints as linear inequalities:

$$\text{min} \quad c^T x \quad \text{s.t.} \quad Ax \geq b, x \in [0,1]^{n(n-1)/2} \tag{3.11}$$

This is a linear relaxation of the TSP, because every solution of (3.11) is also a solution of (3.10), but the contrary need not be true.

Moreover, the cost $c^T x^*$ of a solution x^* of (3.11) is a *lower bound* for the optimum of (3.10).

3.2.1 Separation methods

To apply the branch & cut method to any problem, it is necessary to define for each type of cuts an algorithm which, given the current fractional point $\bar{x}$ and the current LP, finds a constraint that can be added to the LP and which is violated by $\bar{x}$, but is not violated by any feasible solution of the problem.

This kind of procedures is called a *separation method*. Separation methods can be divided into two categories:

- *Exact:* the ones for which there is a proof that there's no $\bar{x}$ that violates some cut for which the procedure was designed and is not found by the algorithm.

- *Heuristic:* there are some $\bar{x}$ that violate some cut for which the procedure was designed and that are not identified.

One can prefer an heuristic procedure over an exact one depending on several factors:

- *Contingental:* there is no known exact separation procedure for that particular family of cuts.

- *Computational:* there is at least one exact separation procedure, which unfortunately is too time-consuming to be of practical interest.

Template paradigm for the TSP

The affine hull $\mathcal{S}$ of all tours on V is given by the solutions x of the continuous relaxation of the two-matching problem.

Any cut is the sum of:

- a linear combination of the degree equations (3.12)
- a non-negative combination of linear inequalities that induce facets of $\mathcal{S}$

These observations lead to a two-phase paradigm for finding cuts for the TSP:

1. describe some family of linear inequalities that induce facets of $\mathcal{S}$
2. find an efficient algorithm that, given a fractional point $\bar{x}$ finds a cut following the template given in phase (1)

In CONCORDE the authors tried to define new separation methods that find cuts that don't respect the template paradigm, called *local-cuts*.

3.2.2 Degree equations (DEG)

Most of the LP representations of the TSP start by inserting a very sparse kind of constraints, the *degree equations*. They enforce a common characteristic of all Hamiltonian tours on a graph, the fact that the tour has to touch any node only once:

$$\sum_{u \in e} x_e = 2 \qquad \forall u \in V \tag{3.12}$$

Where V is the set of all nodes, u is a node and $\{u \in e \Leftrightarrow e \text{ is and extremum of } u\}$.

The number of degree equations is $|V|$, and they present just two non-zero coefficients in any row or column, being very sparse.

The 01-LP minimization problem which included just the degree equations is called the *(exact) two-matching* problem.

Since the degree equations are very sparse and quick to test and add to the LP, they tend to be inserted in the LP from the beginning, without using a separation method. The separation method is anyway straightforward: one just finds any nodes for which the degree equations are violated, then inserts the corresponding constraint.

3.2.3 Subtour-elimination constraints (SEC)

A problem with the solutions obtained using the TSP LP with just the degree equations, is that they generally lead to subtours, i.e. cycles of nodes which are not connected to each other. Subtours can obviously form in any subset $\mathcal{S}, |\mathcal{S}| >= 3, \mathcal{S} \subset V$. A way to forbid any subtour is to insert a *subtour-elimination constraint* for each subset, in the form:

$$\sum_{e=(u,v), u \in \mathcal{S}, v \notin \mathcal{S}} x_e >= 2 \qquad \forall \mathcal{S} \subset V, |\mathcal{S}| >= 3$$

We define for each set $\mathcal{S}$ and for a solution vector x the value

$$\delta_x(\mathcal{S}) = \sum_{e=(u,v),u\in\mathcal{S},v\notin\mathcal{S}} x_e$$

so that the previous constraints can be written more concisely as:

$$\delta_x(\mathcal{S}) \geq 2 \qquad \forall \mathcal{S} \subset V, |\mathcal{S}| >= 3$$

Another complementary way of writing the subtour elimination constraints is to observe that the number of edges that connect vertices in any of the previously defined subsets $\mathcal{S}$ must be at most $|\mathcal{S}| - 1$, in fact if they were more than that, then they would form a cycle in the set

The constraints become:

$$\sum_{e=(u,v),u\in\mathcal{S},v\in\mathcal{S}} x_e \leq |\mathcal{S}| - 1 \qquad \forall \mathcal{S} \subset V, |\mathcal{S}| >= 3$$

and if we define for each set $\mathcal{S}$ and for a solution vector x the value

$$x(E(\mathcal{S})) = \sum_{e=(u,v),u\in\mathcal{S},v\in\mathcal{S}} x_e$$

we can write more concisely:

$$x(E(\mathcal{S})) \leq |\mathcal{S}| - 1 \qquad \forall \mathcal{S} \subset V : 3 \leq |\mathcal{S}| \leq |V|/2)$$

The subtour elimination constraints are facet-inducing, and there are very efficient exact and heuristic separation methods to generate this kind of cuts. Most of the exact separation methods use an algorithm based on finding the *minimum cuts* related to the current fractional point, which will be described in the next chapter, since they form the basis of the *compatible tour heuristic* we studied.

IP Formulation for the TSP

The IP problem defined by:

$$\begin{aligned}
\min \quad & \textstyle\sum_{e\in E} c_e x_e & & \\
\text{s.t.} \quad & x(\delta(\{i\})) = 2 & (i \in V) & \quad (3.13)\\
& x(\delta(\mathcal{S})) \geq 2 & (\mathcal{S} \subset V : 3 \leq |\mathcal{S}| \leq |V|/2) & \quad (3.14)\\
& x_e \in \{0,1\} & (e \in E). & \quad (3.15)
\end{aligned}$$

completely solves the TSP to the optimum.

However, as we stated above, an IP is $\mathcal{NP}$-hard to solve in general.

We can observe that the upper bounds $x_e \leq 1$ implicit in (3.15) can then be viewed as 'degenerate' SECs in which $|\mathcal{S}| = 2$.

Chapter 4

SEP Relaxation, Minimum Cuts and a Tour Construction Heuristic

In this chapter, we define a tour construction heuristic, called the *Compatible-tour heuristic*, which is based on a continuous relaxation of problem (3.13).

4.1 Subtour Elimination Polytope (SEP) Relaxation

To approach the solution of problem (3.13), it is common to *relax* the integrality constraints on the variables, changing it to

$$\begin{aligned} \min \quad & \textstyle\sum_{e \in E} c_e x_e & & \\ \text{s.t.} \quad & x(\delta(\{i\})) = 2 & (i \in V) & \quad (4.1) \\ & x(\delta(S)) \geq 2 & (S \subset V : 3 \leq |S| \leq |V|/2) & \quad (4.2) \\ & x_e \in [0,1] & (e \in E) & \quad (4.3) \end{aligned}$$

We will call the above formulation the *subtour relaxation*. (It is sometimes also called he *Held-Karp relaxation*, because Held & Karp [14] gave an algorithm, based on Lagrangean relaxation, for quickly computing an approximate solution to it.)

We define the polytope induced by the SECs the Subtour Elimination Polytope.

The set of feasible solutions to the subtour relaxation, viewed as a polyhedron in $\mathbb{R}^{|E|}$, is called the *subtour elimination polytope* and, when there are n vertices, denoted by SEP(n) (see, e.g., Boyd & Pulleyblank [6]). More formally,

$$SEP(n) = \left\{ x \in [0,1]^{|E|} : (4.1), (4.2) \text{ hold} \right\}.$$

The subtour relaxation can be solved in polynomial time, for example by the ellipsoid method, but in practice it can be solved very quickly by a linear programming-based cutting plane technique.

4.1.1 Separation method for SECs

An exact separation method for SECs on a fractional point x^* has to find the existing sets $\mathcal{S}$ for which x^* violates the corresponding SEC.

Minimum Cuts

The basis of such a method is finding the minimum value

$$m_{x^*} = \min\left(\delta_{x^*}(\mathcal{S}), (S \subset V : 3 \leq |S| \leq |V|/2)\right)$$

The sets $\mathcal{S}$ for which $\delta_{x^*}(\mathcal{S}) = m_{x^*}$ are called the *minimum cuts* for vector x^*.

This definition is based on the fact that a *cut* $\mathcal{C}(\mathcal{S})$ on a graph $G = (V, E)$ is the set of all the edges in E which cross the boundary of $\mathcal{S} \subset V$. The cost of the cut if we apply x^* as the cost vector of the graph is the sum of all the edges in the cut, or $\delta_{x^*}(\mathcal{S})$.

The algorithm for finding violated SECs for vector $\bar{x}$ is then:

```
1 Find the minimum cut value m_x̄
2 if  m_x̄ < 2 then
3     find all minimum cuts S in G for x̄
4 end
```

Algorithm 3: SEC separation

All the sets $\mathcal{S}$ found by algorithm (3) violate the subtour elimination constraints, so to solve the SEP relaxation of the TSP we can iterate it and write:

```
1  Find the minimum cut value m_x̄
2  if m_x̄ ≥ 2 then
3      STOP
4  else
5      if m_x̄ < 2 then
6          find all minimum cuts S in G for x̄
7          and add a SEC for each one of them
8          then reoptimize with dual simplex
9      end
10 end
```

Algorithm 4: Optimizing over the SEP

At the end of algorithm (4) the minimum cut value for solution x^* is 2.

4.2 A Tour Construction Heuristic

In this work we present the Compatible Tour Heuristic for the TSP, which can be viewed as a *tour construction* heuristic, i.e. one that constructs a tour for the TSP based on some additional rules.

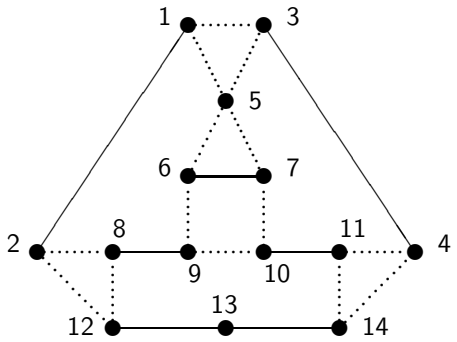

Figure 4.1: A fractional vertex of SEP(14).

An example of tour construction heuristic is the Christofides algorithm, which is based on finding the shortest spanning-tree on graph G, extending it to solve a perfect matching, doubling the resulting graph, finding an eulerian tour on the graph and shortcutting it to get the solution.

4.2.1 The Compatible Tour Heuristic

We denote by x^* the solution to the subtour relaxation. Without loss of generality, we assume that x^* is a basic solution (i.e., it corresponds to a vertex of the subtour polytope).

The key to our heuristic is the notion of *tight sets*:

Definition: We say that a vertex set $S \subset V$ is *tight* at x^* if $x^*(\delta(S)) = 2$.

Intuitively, if a set is tight at x^*, there is a high probability that it will also be tight in the optimal tour. (This intuition has been confirmed in our computational experience.) This motivates the following definition:

Definition: We say that a tour T is *compatible* with x^* if the associated incidence vector ($\bar{x}$, say) has the following property: every vertex set which is tight at x^* is also tight at $\bar{x}$. (The reverse need not be true.)

Figures 4.1 and 4.2 illustrate this concept. Figure 4.1 shows a fractional vertex of SEP(14). Solid lines represent variables with value 1, dotted lines represent variables with value $1/2$. The sets $\{1, 2\}$, $\{6, 7\}$ and $\{5, 6, 7\}$, for example, are tight. Figure 4.2 shows a tour compatible with this fractional vertex.

The compatible tour heuristic for the STSP is as follows: optimize over the subtour polytope, giving a solution vector x^*, and then find the best tour compatible with x^*. Of course, it is not obvious how to find the best compatible tour, so that some thought has to be put into how to actually implement the heuristic. (The number of compatible tours can be exponential in n.)

Since the heuristic does not use any local search, one might categorise it as a 'constructive' heuristic. Yet, it differs from most constructive heuristics (such as the

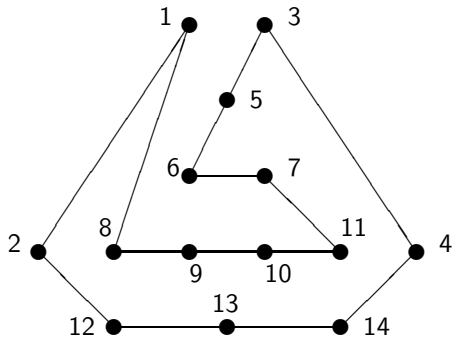

Figure 4.2: A tour compatible with the fractional vertex shown in Figure 4.1.

'double spanning tree' heuristic of Rosenkrantz, Stearns & Lewis [16] or the 'spanning tree plus matching' heuristic of Christofides [8]), in that the tour is selected from a well-defined, exponentially-large collection of tours. Some other heuristics of this type have been proposed recently, such as the 'double spanning tree with optimal shortcuts' heuristic of Deineko & Tiskin [9] and the heuristics based on triangulations described in Letchford & Pearson [92].

Chapter 5

Two Negative Results

In this chapter, we prove two negative results. The first is that finding the best compatible tour is a hard problem. For this, we need the following well-known result.

Proposition 1 (Garey, Johnson & Stockmeyer [12])**.** *Testing if a cubic (i.e., 3-regular) graph is Hamiltonian is $\mathcal{NP}$-complete in the strong sense.*

We will also need the following lemma.

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

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

Example: The cubic graph G on the left of Figure 5.1 is Hamiltonian. A suitable set C consists of the edges $\{1, 2\}$, $\{2, 3\}$, $\{3, 4\}$, $\{4, 5\}$, $\{5, 6\}$ and $\{6, 1\}$. The resulting multigraph G' is displayed on the right of the figure. There are three edge pairs that need to be inserted in the sequence. The pair $\{2, 4\}$, $\{4, 2\}$ is inserted between $\{1, 2\}$ and $\{2, 3\}$, the pair $\{3, 6\}$, $\{6, 3\}$ is inserted between $\{2, 3\}$ and $\{3, 4\}$, and the pair $\{1, 5\}$, $\{5, 1\}$ is inserted between $\{6, 1\}$ and $\{1, 2\}$. The resulting traversal of G' is $\{1, 2\}$, $\{2, 4\}$, $\{4, 2\}$, $\{2, 3\}$, $\{3, 6\}$, $\{6, 3\}$, $\{3, 4\}$, $\{4, 5\}$, $\{5, 6\}$, $\{6, 1\}$, $\{1, 5\}$, $\{5, 1\}$.

We are now ready to prove the hardness result.

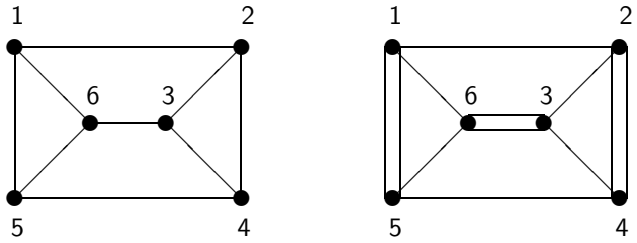

Figure 5.1: Cubic Hamiltonian graph G and Eulerian multigraph G'.

5.1 Finding the best compatible tour is $\mathcal{NP}$-hard in the strong sense.

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

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

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

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

Figure 5.2 illustrates the $1/2$-integral solution corresponding to the cubic biconnected graph illustrated in Figure 5.1. Solid (respectively, dotted) lines represent edges whose variables have value 1 (respectively, $1/2$).

Note that each of the $3n/2$ edges of zero cost forms a tight set. Any compatible tour must traverse these $3n/2$ edges. Now, note that, since all vertex degrees are odd in a cubic graph, any compatible tour must use at least $n/2$ of the expensive edges (i.e., those of cost M or M^2). In order to minimize costs, the optimal compatible tour will use, if possible, no edges of cost M^2 and exactly $n/2$ edges of cost M. Such a tour also uses exactly n edges of cost 1, and therefore has a total cost of $n(1 + M/2)$. (Figure 5.3 illustrates such a compatible tour for our example. The 3 dotted lines represent the edges of cost M.)

Now we show that such a compatible tour exists if and only if G is Hamiltonian. First, if G is Hamiltonian, then, by Lemma 1, for any tour there exists a traversal

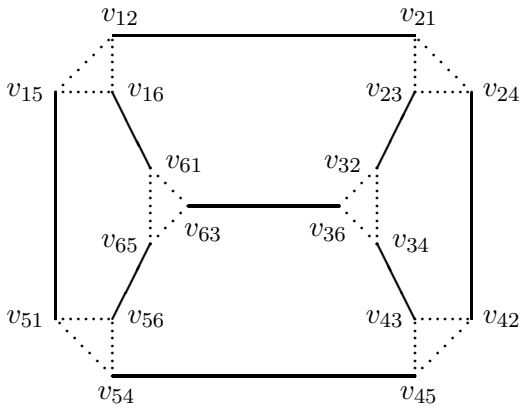

Figure 5.2: Extreme point of SEP(18).

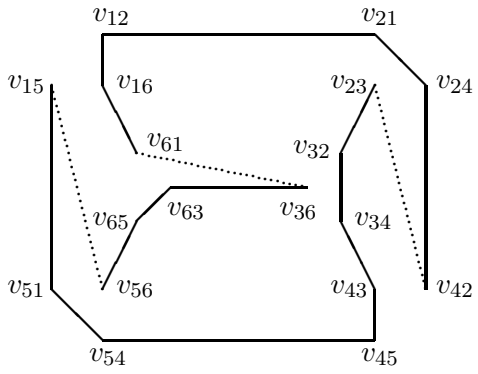

Figure 5.3: Compatible tour corresponding to traversal of G'.

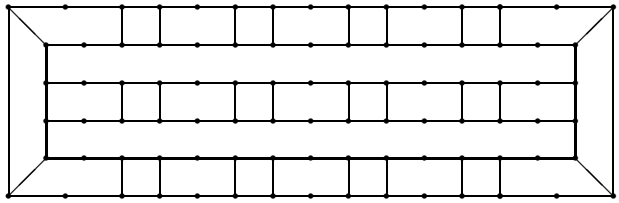

Figure 5.4: The graph H_{35}.

of G' in which the edges are visited in the same order that they are visited in the Hamiltonian circuit itself. Such a traversal can be extended to a compatible tour of the desired form for the STSP instance. On the other hand, if there exists a compatible tour of the desired form, it corresponds to a traversal of G' that can then be 'short-cut' (by omitting the edge-pairs in the duplicated matching) to yield a Hamiltonian circuit of G. □

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

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

5.2 The worst-case ratio versus the optimal tour is at least $5/3$

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

Proof. We construct a family of instances as follows. Let $p \geq 2$ and $q \geq 2$ be arbitrary positive integers. We construct a graph H_{pq} on $6pq$ vertices as follows (see Figure 5.4 for an illustration). For $i = 1, \ldots, pq-1$ and $j = 0, \ldots, 5$, vertex $jpq+i$ is connected to vertex $jpq+i+1$ by an edge. For $i = 0, 1, 2$ and $j = 0, \ldots, q-1$, vertex $2ipq + jq + 1$ is connected to vertex $(2i+1)pq + jq + 1$ and vertex $2ipq + (j+1)q$ is connected to vertex $(2i+1)pq + (j+1)q$. Finally, vertex 1 is connected to vertex $5pq + 1$ and vertex pq is connected to vertex $6pq$.

Associated with the graph H_{pq}, we define an STSP instance with $6pq$ vertices as follows. If the edge $\{i, j\}$ appears in H_{pq}, then the cost of that edge is set to 1 in the STSP instance. Otherwise, the cost is set equal to the number of edges in the shortest path from i to j in H_{pq}. The resulting costs clearly satisfy the triangle inequality.

It is easy to check that several Hamiltonian circuits exist in the graph H_{pq}. One such circuit is shown in Figure 5.5 for H_{35}. Since each edge used in such a tour has a

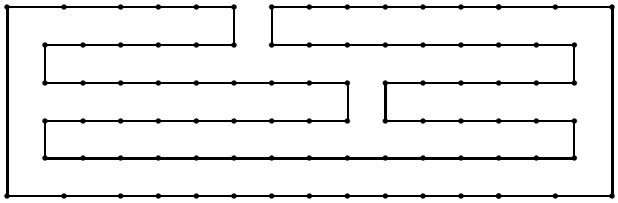

Figure 5.5: A Hamiltonian circuit in H_{35}.

Figure 5.6: Optimal fractional vertex of SEP(90).

cost of 1, each of these circuits represents an optimal solution of the STSP instance with a cost of $6pq$.

Since every point in SEP($6pq$) satisfies $x(E) = 6pq$, and every edge has a cost of at least 1, the subtour lower bound is equal to the cost of the optimal tour(s) for this family of instances. That is, any optimal tour is also an optimal solution to the subtour relaxation. Moreover, there exist many alternative optimal solutions to the subtour relaxation that are fractional. One such fractional solution is shown in Figure 5.6. As before, solid (respectively, dotted) lines represent edges e with $x_e^* = 1$ (respectively, $x_e^* = 1/2$). There are $6(p-1)q$ edges with $x_e^* = 1$ and $12q$ edges with $x_e^* = 1/2$. Since all of these edges have cost 1, the cost of the fractional solution is also equal to $6pq$ as stated.

It is a simple exercise to show that the cheapest tour compatible with the fractional point of the form displayed in Figure 5.6 has a cost of more than $(10q-2)p$ (See Figure 5.7 for an illustration.) As p and q approach infinity, the ratio of $(10q-2)p$ to $6pq$ approaches $5/3$. □

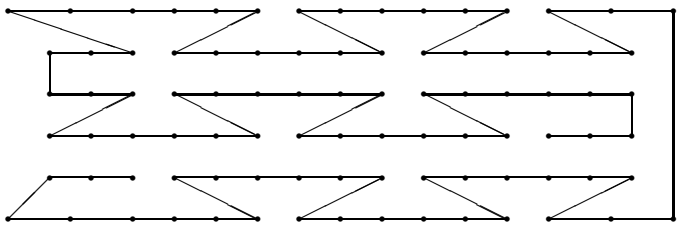

Figure 5.7: Compatible tour for H_{35} (one edge omitted for clarity).

Chapter 6

Algorithms

Despite the negative complexity result, there are some algorithms for computing the best compatible tour that work well on many instances of practical interest. In this section, we describe two such algorithms. These algorithms are based on some structural results about tight sets, that we establish next.

6.1 Tight sets

Given a point x^* belonging to the subtour polytope, the *support graph* $G^* = (V, E^*)$ is the subgraph of G induced by edges whose variables are positive at x^*, that is, $E^* = \{e \in E : x^*_e > 0\}$. As usual, for any edge $e \in E^*$, we interpret the variable value x^*_e as the *weight* of e. Note that, under the assumption that x^* satisfies all SECs, each tight set $S \subset V$ induces a minimum weight cut in G^*. In this section, we recall some structural properties which are known to hold for the tight sets of G^*, as a direct consequence of known results for minimum cuts in weighted graphs.

The following result is well known, but we give a proof for the sake of completeness.

Proposition 2. *Suppose S_1 and S_2 are two vertex sets which* cross, *i.e., that $S_1 \setminus S_2$, $S_2 \setminus S_1$, $S_1 \cap S_2$ and $V \setminus (S_1 \cup S_2)$ are all non-empty. If S_1 and S_2 are tight, then so are $S_1 \cap S_2$ and $S_1 \cup S_2$.*

Proof. By definition,

$$x(\delta(S_1)) + x(\delta(S_2)) \geq x(\delta(S_1 \cap S_2)) + x(\delta(S_1 \cup S_2)). \tag{6.1}$$

Since S_1 and S_2 are assumed to be tight, the LHS of (6.1) is $2 + 2 = 4$. Since x^* is assumed to satisfy all SECs, this immediately implies that $x(\delta(S_1 \cap S_2)) = x(\delta(S_1 \cup S_2)) = 2$. □

This property leads naturally to the concept of a *necklace*.

Definition 1. *A* necklace *is a partition of V into subsets $S_1, \ldots, S_m (m \geq 3)$ such that the union $\bigcup_{k=i}^{k=i+j} S_k$ is tight for any $1 \leq i \leq m$ and any $0 \leq j < m - 1$ (indices taken modulo m).*

It is assumed that each necklace defines a partition which is as fine as possible, i.e., that no tight set S_i in the necklace can be partitioned further into two or more smaller tight sets. Following Applegate et al. [17], we call the individual tight sets S_i the *beads* of the necklace, and we call the union of two adjacent beads, i.e., a tight set of the form $S_i \cup S_{i+1}$, a *domino*. For example, for the fractional point shown in Figure 4.1, one necklace has the beads $\{1, \ldots, 11\}$, $\{12\}$, $\{13\}$ and $\{14\}$. The associated dominoes are therefore $\{1, \ldots, 12\}$, $\{12, 13\}$, $\{13, 14\}$ and $\{1, \ldots, 12\} \cup \{14\}$.

It is also necessary to allow 'degenerate' necklaces in which $m = 2$. These simply represent vertex sets S (and their complements $V \setminus S$) which are tight, but which cannot be viewed as part of a larger necklace. For the point shown in Figure 4.1, the partition of V into $\{1, 2\}$ and $\{3, \ldots, 14\}$ forms a degenerate necklace.

Proposition 3. *The tight sets for a given x^* can be arranged in $\mathcal{O}(n)$ necklaces. Moreover, the total number of beads and dominoes over all necklaces, including degenerate ones, is also $\mathcal{O}(n)$.*

Corollary 1. *The total number of tight SECs is $\mathcal{O}(n^2)$.*

The concept of necklaces provides a nice representation of the tight sets. However, there exists a more compact (linear-space) data structure for representing and storing the tight sets: the *cactus* of Dinitz, Karzanov & Lomonosov [10]. Later, Applegate et al. [17] showed that one can instead use the *PQ-tree* of Booth & Lueker [5]. The two representations are equivalent.

6.1.1 The cactus-tree representation of minimum cuts

As we said the cactus-tree representation of minimum cuts in a graph is more compact than the original set of minimum cuts. A cactus tree can be stored in fact in $\mathcal{O}(n)$ space: more precisely, it has at most $2n$ nodes and $\mathcal{O}(n)$ edges.

The *cactus* is defined as a weighted graph in which every edge is in at most one cycle. There are two types of edges:

- *cycle edges* are in one cycle
- *tree edges* are in none. Tree edges have twice the weight of cycle edges.

For a cactus we say node instead of vertex. Any node in a cactus maps to a – possibly empty – set of vertices of the original problem.

We define:

- an *empty node* as a node that doesn't map to any vertex in the original problem.
- a *k-way cut node* as a node which, when removed, splits the graph in k connected components.

To extract a minimum cut from a cactus, one has to remove two different edges in a cycle, or one tree edge. This operation splits the cactus in two connected components. The minimum cuts are found by collecting the vertices contained in the cactus' nodes for one of these two connected components.

Every cycle in the graph forms a necklace. Removing every edge in a cycle with k edges leaves k connected components which are the *beads* of the necklace.

A degenerate necklace is obtained from:

- tree nodes: they represent a (trivial) mincut made by a single node
- 2-way cutnodes

Cacti of TSP Support Graphs

When a cactus is generated from a fractional point in SEP, for which every single vertex is a mincut, it has some added properties:

- every non-empty node in the cactus maps to just one vertex in the TSP
- every cutnode is empty

6.1.2 PQ-trees

A PQ-tree is a rooted acyclic tree composed of P-nodes, Q-nodes and leaf-nodes. Each leaf-node corresponds to a vertex of the original graph. A P node has at least two children, and a Q node has at least three children.

A P-node indicates that, when traversing the graph, its descendants can be visited in any order.

A Q-node forces to visit its descendants consecutively, i.e. in one of the two possible orders: left-to-right or right-to-left.

It is possible to construct a PQ-tree from a TSP support graph, so that removing any edge splits the tree in two connected components, which represent a minimum cut and its complement.

PQ-trees are equivalent to cactus-trees, and they can express the structure of the minimum cuts of a graph in a compact and efficient way.

Figure 6.1 shows the PQ-tree corresponding to the point shown in Figure 4.1. Following Applegate et al., the circle denotes a P-node and the squares denote Q-nodes. The P-node expresses the fact that the vertex sets $\{1, 2\}$, $\{3, 4\}$, $\{5, 6, 7\}$, $\{8, 9\}$, $\{10, 11\}$ and $\{12, 13, 14\}$ are tight, but that the union of any two of these sets is not tight. (Equivalently, each of these six sets is part of a degenerate necklace.) The Q-node on the bottom right, for example, expresses the fact that the partition of V into vertex sets $\{1, \ldots, 11\}$, $\{12\}$, $\{13\}$ and $\{14\}$ defines a necklace. There are seven Q-nodes for this example, meaning seven necklaces in total. Figure 6.2 shows the cactus tree corresponding to the same point. The nodes with number greater than 14 are the empty ones.

The conversion between a cactus-tree and a PQ-tree is clearly straightforward, even if not unique:

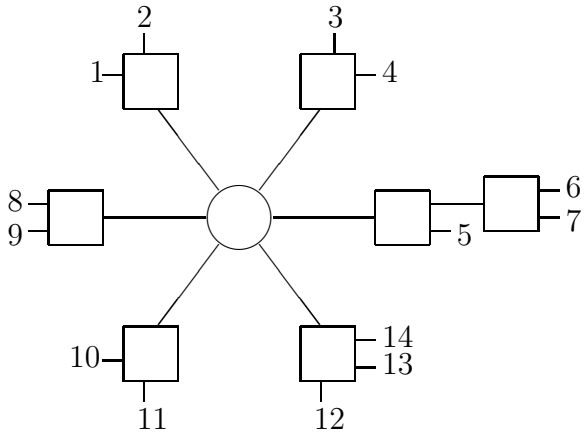

Figure 6.1: PQ-tree corresponding to the fractional point in Figure 4.1.

- Any k-way cutnode with $k \geq 4$ is a P-node
- Any cycle is a Q-node

For a given point x^*, the cactus tree or PQ-tree can be computed efficiently, for example in $\mathcal{O}(nm \log(n^2/m))$ time using the algorithm of Fleischer [11] or in $\mathcal{O}(nm \log n \log(n^2/m))$ time using the algorithm of Wenger [1]

6.1.3 Dominance among Tightness Constraints

At this point it is useful to restate Proposition 2 in terms of linear inequalities:

Lemma 2. *Suppose S_1 and S_2 are two vertex sets which* cross. *The equations $x(\delta(S_1)) = 2$ and $x(\delta(S_2)) = 2$, together with non-negativity, imply the equations $x(\delta(S_1 \cap S_2)) = 2$ and $x(\delta(S_1 \cup S_2)) = 2$.*

The above structural results can then be used to characterise the compatible tours, in two different ways.

Proposition 4. *A tour $\bar{x}$ is compatible with x^* if and only if:*

- $\bar{x}(\delta(S_i \cup S_{i+1})) = 2$ *for all dominoes $S_i \cup S_{i+1}$, and*
- $\bar{x}(\delta(S_i)) = 2$ *for each tight set belonging to a degenerate necklace.*

Corollary 2. *Using the alternative expression of the SECs, the first constraint in proposition 4 can be rewritten as:*

$$\bar{x}(E(S_i \cup S_{i+1})) = |S_i \cup S_{i+1}| - 1 \text{ for all dominoes } S_i \cup S_{i+1}$$

Proposition 5. *A tour $\bar{x}$ is compatible with x^* if and only if:*

- $\bar{x}(\delta(S_i)) = 2$ *for all beads S_i, and*

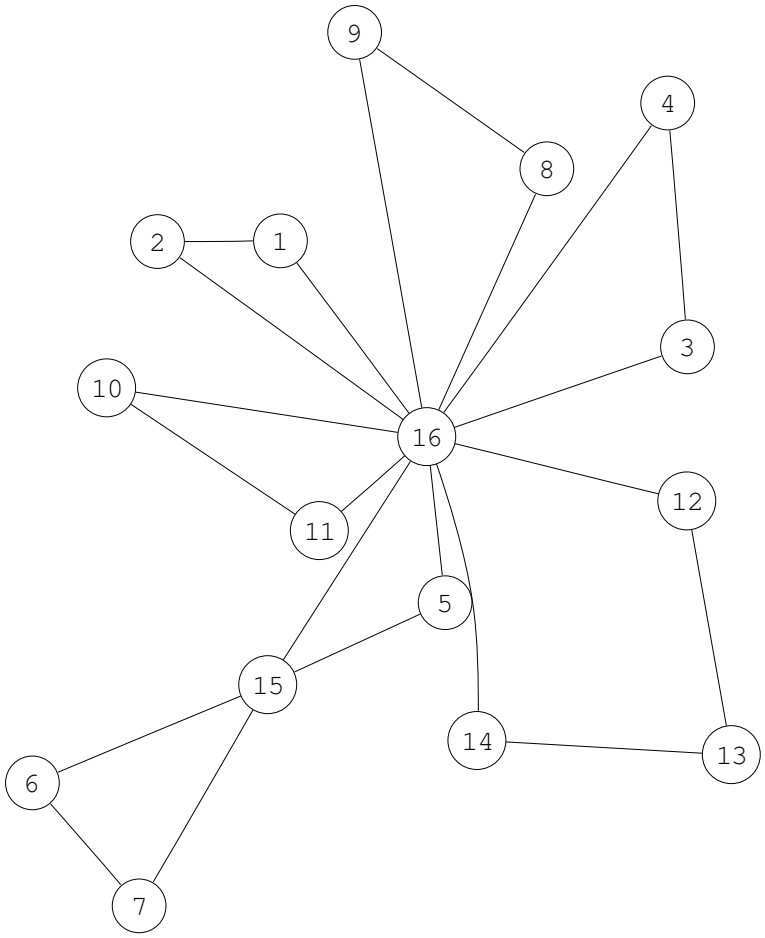

Figure 6.2: Cactus-tree corresponding to the fractional point in Figure 4.1 (nodes above 14 are empty)

- $\bar{x}_{uv} = 0$ *for each edge* $\{u, v\}$ *such that* u *and* v *appear in non-consecutive beads of a necklace.*

Proof. Let B_1, B_2 and B_3 be 3 consecutive beads in a necklace, and let domino D_1 be composed of B_1 and B_2 and domino D_2 be composed of B_2 and B_3. Then we have:

$$x(\delta(D_1)) + x(\delta(D_2)) = x(\delta(B_1)) + x(\delta(B_3)) + 2x(B_2 : (V \setminus (B_1 \cup B_2 \cup B_3))).$$

But B_1 and B_3 are tight and $x(B_2 : (V \setminus (B_1 \cup B_2 \cup B_3)))$ is zero by construction. Hence $(\delta(D_1)) + x(\delta(D_2)) = 4$, which, if SECs and non-negativity hold, implies that D_1 and D_2 are also tight. □

These results are of interest because, when looking for the best compatible tour, we will need to somehow enforce that all sets which are tight at x^* be also tight in the tour. A natural way to do this in the linear programming context is to add one equation for each of the $\mathcal{O}(n^2)$ tight SECs, but, as the above propositions show, $O(n)$ equations suffice. The latter method has the advantage that, as well as adding only $O(n)$ equations, one also gets to eliminate some edges entirely.

Using this latter method, it is not difficult to see that:

Corollary 3. *It is always possible to choose the set of* $O(n)$ *equations in such a way that the corresponding tight cuts form a laminar family.*

6.1.4 Properties of a compatible tour

Proposition 6. *From proposition 2 follows that all the edges in the cactus tree that connect two non-empty nodes will be in the compatible tour.*

Proposition 7. *If a cycle has* $n > 3$ *beads, then the beads are visited in the order set by the cycle.*

6.2 A dynamic programming algorithm

6.2.1 PQ-tree based algorithm

Recall from the previous section that the cactus tree or PQ-tree provides a compact ($\mathcal{O}(n)$-sized) representation of the tight sets. Using the ideas of Burkhard, Deineko & Woeginger[7], there is a combinatorial algorithm (based on dynamic programming) to find the best compatible tour. The algorithm runs in polynomial time when the degree of each P-node in the PQ-tree (or, equivalently, each cut-node in the cactus tree) is bounded by a constant, but in exponential time if P-nodes of arbitrary degree are allowed. Unfortunately, we have the following negative result:

Theorem 3. *PQ-trees associated with extreme points of the subtour polytope can contain P-nodes of arbitrarily large degree.*

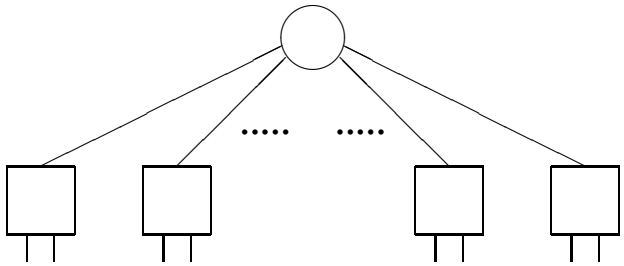

Figure 6.3: PQ-tree corresponding to the fractional point in Figure 5.2.

Proof. Consider again the proof of Theorem 1 and assume that the biconnected cubic graph G is also 3-edge-connected. Then, in the fractional point for the associated TSP instance, all of the tight sets have cardinality 2. Indeed, there are precisely $|E|$ such tight sets, one for each edge of E. (Since G is 3-edge-connected, all other vertex sets $S \subset V'$ satisfy $x^*(\delta(S)) \geq 3$.) Therefore, the PQ-tree contains a single P-node, of degree $|E|$, and $|E|$ Q-nodes (see Figure 6.3.) Since biconnected, cubic, 3-edge-connected graphs can be arbitrarily large, the result follows. □

Fortunately, in practice the degrees of the P-nodes (if any) tend to be very small. In 20 instances from TSPLIB, there is only one instance in which the maximum degree exceeds 4. In that case, it is 6. So we would expect the dynamic programming algorithm to be fast in practice (certainly much faster than solving the original TSP instance to optimality via branch-and-cut).

Another observation is that, when looking for the best compatible tour, we can use reduced costs instead of original costs. It is not clear whether this could be exploited.

6.2.2 Cactus-tree based algorithm

The dynamic programming algorithm we developed for solving the compatible tour problem is based on the following result:

Proposition 8. *From any eulerian tour $\mathcal{E}$ on the cactus we can derive a compatible tour by just eliminating the* empty nodes *of the cactus.*

An *eulerian tour* for a graph $G = (V, E)$ is a tour that uses all the edges in E. Such a tour always exists on a cactus tree, since every node has an even number of incident edges.

For a cactus $C = (W, F)$ of a TSP with graph $G = (V, E)$, we define the ordered pair

$$P_u\,(S, v)\,, \text{ where } S \subseteq V, v \in S$$

as the minimum Hamiltonian Path through all the nodes in S, which

- uses any edge in F at most once
- starts from node $u \in S$ and ends in v.

We then use the following property:

$$P_u(S, v) = \min(\{P_u(S \setminus \{w\}, w)\})$$

and define

$$P_u(\{v\}, v) = 0 \qquad \forall v \in V$$

Then the shortest compatible tour is given by

$$T = \min(P_u(V, v) + c_{v,u})$$

If V_f is the set of vertices in W which are extrema of edges in $(u, v) \in F$ with both $u \in V$ and $v \in V$, then the first set of initial solutions for the DP-algorithm we used is:

$$\{P_u(\{u, v\}, v) = c_{u,v}\}, u, v \in V_f$$

And there are obviously at most 2 such sets. Due to the cyclic nature of the cactus, we can just choose one of the two and start from just one set.

Then from any subproblem $P_u(S, v)$ we found the next generation subproblem as $P_u(S \cup \{w\}, w)$ by extending it using a neighbor $w \in V$.

The search for a neighbor node is done in this way:

- Let F' be the set $\{(u, v) \in W, u$ is empty OR v is empty$\}$
- For each subproblem $P_u(S, v)$ we save the graph F'_{P_u} composed of the remaining edges in F' for P_u, i.e. the ones that still have to be used by P_u.
- To determine the edges used by a subproblem, we say that if $P'_u(S \cup \{w\}, w)$ derives from $P_u(S, v)$ by extending it from node v to node w,
  - if edge $(v, w) \in F$, then $F'_{P_u} \equiv F'_{P'_u}$
  - otherwise, we define the used edges for (v, w) as the shortest path from v to w in F'_{P_u}, with all edges' costs set to 1.

  If such a shortest path doesn't exist, then we can't extend $P_u(S, v)$ with w.

Then a node w is a neighbor to v for problem $P_u \iff$ it is reachable from v on graph G_{P_u}.

6.2.3 Improvements on the algorithm

The algorithm described above can be largely improved by exploiting some properties of the compatible tour, related to the cactus.

First of all, we can use proposition 6 to derive

Corollary 4. *Define a* one-chain *as a hamiltonian path on cactus C in a set $O \subseteq V$ that starts with a node $u \in O$ and ends with node $v \in O$, passing through all nodes in V. (The one-chain derives its name by the fact that it is composed of an uninterrupted sequence of edges at the upper bound in the fractional point from which the cactus is derived.)*

The compatible tour is a cyclic connection of all one-chains in C.

All one-chains $(u, \ldots, v)$ can then be shrunk, i.e. substituted by a single edge (u, v) with cost equal to the length of the path from u to v.

If U is the set of one-chains, then we can change the initial set of subproblems to

$$\{P_u(\{u, v\}, v) = c_{u,v}\}, (u, v) \in U$$

And we observe that there is just one such set for any starting node after shrinking the graph.

Another property is that the set of neighbors w to test for a node v can be reduced –in general– by applying property (7):

- if we are trying w as a neighbor to v in subproblem $P_u(S, v)$
- if we eliminate from the remaining edges F'_{P_u} of $P_u(S, v)$ the path from v to w, getting $F'_{P'_u}$
- if the set of nodes reachable from v on the $F'_{P'_u}$ is not the same as the set of nodes that was reachable from w on F'_{P_u},

then going from v to w would mean that the resulting subproblem would be pruned in the future iterations, since it would have no neighbors to its last node, without having covered all the nodes in V.

A new implementation using all these improvements leaded to a much smaller number of subproblems to be tested.

Lower bound

During the exploration of the subproblems, we can substitute the length of the path $\lambda_{u,v}$ in S from u to v by the projected length of the solution, adding to $\lambda_{u,v}$ the length of all the one-chains that have still to be inserted in P_u.

Early pruning

We can observe that it is very straightforward to find a complete compatible tour by exploring the subproblems' structure in depth-first order, by repeatedly extending a starting subproblem.

It is thus possible to find one or more solutions with depth-first exploration, keeping the best one t as an upper bound and then pruning any subproblem which lower bound is greater or equal to the cost of t

6.3 LP-based implementations

A direct implementation of the compatible tour heuristic is to directly insert in the subtour LP the compatibility constraints. This is the same of changing to equality all the SECs that are tight for x^*.

All the approaches we present are based on first solving the subtour relaxation, then finding the cactus tree representation of the tight sets in the resulting extreme point x^*.

From the results presented in proposition 4, for each non-degenerate necklace $S_1, S_2, \ldots, S_m$ we add the constraints:

$$\begin{aligned} \sum_{i \in D_k, j \notin D_k} x_{i,j} &\leq 2 \text{ for each } D_k \text{ in } S_k \cup S_{k+1}, k = 0, \ldots, m-1 \\ \sum_{i \in D_m, j \notin D_m} x_{i,j} &\leq 2 \text{ for } D_m = S_m \cup S_1 \end{aligned} \tag{6.2}$$

and for each degenerate necklace S_1, S_2 we add the constraint:

$$\sum_{i \in S_1, j \notin S_1} x_{i,j} \leq 2$$

which, together with the SECs, force all the tight sets to remain as such.

Moreover, we fix:

- at 1 all the edges that are in a domino of two vertices
- at 0 all the edges that don't connect any two consecutive beads in any necklace.

The CONCORDE TSP solver

We based a large part of our code on the CONCORDE callable library, developed by Applegate, Bixby, Chvátal & Cook for the homonymous TSP solver [2].

It is the most advanced software for the solution of the TSP, and it was able to solve most of the problems that Gerhard Reinelt collected in TSPLIB [3], up to a tour of Germany composed of 15,112 cities.

CONCORDE's callable library offers LP-based functions for the separation of cuts and even for the complete solution of TSP problems. It was very convenient having such a library to avoid reimplementing separation procedures from scratch.

6.3.1 Lagrangean relaxation of the tightness constraints

As a preliminary approach, we tested the heuristic on the CONCORDE software by modifying the original TSP instance after having created the cactus representation of the tight sets.

Since the LP formulation for the TSP we are using (3.13) is a minimization problem, the lagrangean relaxation of the tightness constraints in (6.2) must increase the objective function when any of the constraints is violated.

A constraint of the type

$$\bar{x}(\delta(S)) \leq 2$$

is violated if

$$\bar{x}(\delta(S)) > 2$$

or equivalently if $\bar{x}(\delta(S)) - 2 > 0$

For each domino or degenerate bead S, we add the quantity

$$\lambda\left(\bar{x}(\delta(S)) - 2\right)$$

to the objective function. This is the same as adding λ to the cost $c_{i,j}$ of each edge that crosses the boundary of S.

To speed up the calculations, we can polarize the problem even more by using the dual representation of the SECs, which we showed in (3.2.3): the equation that represents the tightness constraint in the alternative form is

$$\bar{x}(E(S)) \geq |S| - 1$$

which is violated when

$$\bar{x}(E(S)) < |S| - 1$$

or equivalently $(|S| - 1) - \bar{x}(E(S)) > 0$. We can then add for each domino or degenerate bead S the quantity

$$\lambda\left((|S| - 1) - \bar{x}(E(S))\right)$$

to the objective function. This is the same of subtracting λ to the cost $c_{i,j}$ of each edge that is inside S. In the latter case, we must choose λ so that the cost of any edge doesn't become negative, so

$$\lambda \leq \epsilon + \min_{i \in S, j \in S} \{c_{i,j}\}$$

The final algorithm is:

find the SEP extremum point x^* using CONCORDE
for *each domino or degenerate bead* $\mathcal{S}$ **do**
    subtract λ to the cost $c_{i,j}$ of every edge $e_{i,j} : i \in \mathcal{S}, j \in \mathcal{S}$ and add $\lambda\left(|\mathcal{S}| - 1\right)$ to the objective function
    add λ to the cost $c_{i,j}$ of every edge $e_{i,j} : i \in \mathcal{S}, j \notin \mathcal{S}$ and subtract 2λ from the objective function
    solve the modified instance using CONCORDE
end
if *the resulting tour* T *is not compatible* **then**
    increase λ and regenerate the modified instance
else
    return T
end

Algorithm 5: Lagrangean relaxation

The parts of the algorithm that created the new instance were coded in the *Python* programming language, allowing for rapid prototyping.

6.3.2 A branch-and-cut algorithm

A stricter way of implementing the compatible tour heuristic is to add the tightness constraints (6.2) and (6.3) directly to the TSP LP.

At first, we tried to work inside the CONCORDE software for solving the TSP, to modify it in order to find a compatible tour. This approach should the most natural one, since CONCORDE is to date one of the most advanced and performing softwares for solving the TSP. We forced the cutting stage of the root node of CONCORDE's branch-and-cut algorithm to use only the subtour elimination constraints. In this way, at the end of the root node we had the SEP extremum x^*.

After that, we calculated the cactus representation of the root node using Wenger's code, and started enumerating the necklaces.

The problem with inserting the compatibility constraints in CONCORDE, is that they cannot be represented in CONCORDE's cuts' standard format, which uses *hypergraphs* or ordered couples (V, F) where V is a set of vertices and F is a set of (not necessarily disjoint) non-empty subsets of V. This structure is capable of expressing cuts of the form $\sum a_{i,j}x_{i,j} >= b_i$, but not of the form $\sum a_{i,j}x_{i,j} < b_i$. In other words, cuts explicited in this way are valid for any subproblem in the TSP branch-and cut tree: this allows CONCORDE to keep a *pool* of all the cuts it generated, to look into for finding violated ones.

Unfortunately this means that it's not possible to represent in CONCORDE's format any cut that restricts the solution to some subset of the TSP polytope, i.e. forces the solution of some subproblem. CONCORDE of course has a very powerful branching engine, and we implemented the compatibility cuts as subproblems. However, since branching for CONCORDE involves the generation of a whole new instance of the TSP, together with a new bound and a set of edges, adding all the constraints we needed proved to be very time-consuming, up to the point of being impractical except for very small-sized instances.

We then decided to switch to a complete implementation of a branch-and-cut (B&C) algorithm for the compatible tour, in order to test it against more instances.

Implementing the compatible heuristic by B&C with Pricing

The code we wrote to solve the compatible tour heuristic is based on a branch-and-cut with pricing algorithm.

Initial set of edges First of all, it determines a set of edges to insert into the initial LP, which is the union of:

- All the edges in a computed greedy solution
- The first n shortest edges adjacent to every vertex in the TSP

We add all the edges in a solution because this way any cut that is globally valid will have at least a set of edges that can fulfill it. The second set is instead one of the various heuristics that can be used to find a good starting set. The number n of shortest edges can be constant or related to the size of the problem.

The root node The initial LP contained just the DEG equations. The root node is then solved to optimality using a B&C with Pricing algorithm:

```
1 Solve the initial LP
2 repeat
3     repeat
4         Find violated SECs
5         Add violated SECs to the LP and solve with Dual Simplex
6     until There are no violated SECS ;
7     Find variables with negative reduced cost
8     Add variables with negative reduced cost to the LP and solve with Primal
      Simplex
9 until There are no variables with negative reduced cost ;
```

Algorithm 6: Pricing

The algorithm above can be improved by applying these two rules:

- At step 5, add only the m most violated SECs, to try to improve the solution without increasing too much the size of the *tableau*
- At step 8, add only the p most negative variables, for the same reason

Moreover, we applied cut and variable aging:

- For every cut, we keep a counter of the number of optimization cycles (age) in a row in which the cut is not tight
- For every edge, we keep a similar counter for the number of cycles (age) in a row in which the edge has positive reduced costs (i.e. is not in the basis)
- We remove a cut or an edge when its age is greater than some threshold.

The improvements above allow to keep the size of the tableau limited and speed up the computation.

Chapter 7

Results

In this chapter we present some of the results obtained by applying our different approaches to the problem.

7.1 Gap for problems in TSPLIB

In table 7.1 we show the gap associated to some instances of the TSPLIB.

The columns show the instance name, the compatible tour value, the best tour value and the relative gap:

$$gap = \frac{compat - opt}{opt} \times 100$$

Instance	Compatible	TSP Optimum	GAP%
burma14	3323	3323.00	0.00
ulysses16	6859	6859.00	0.00
gr17	2085	2085.00	0.00
gr21	2707	2707.00	0.00
ulysses22	7013	7013.00	0.00
gr24	1328	1272.00	4.22
fri26	937	937.00	0.00
bays29	2039	2020.00	0.93
bayg29	1610	1610.00	0.00
dantzig42	699	699.00	0.00
swiss42	1273	1273.00	0.00
att48	10653	10628.00	0.23
gr48	5226	5046.00	3.44
hk48	11525	11461.00	0.56
eil51	448	426.00	4.91

continued on next page

continued from previous page

Instance	Compatible	TSP Optimum	GAP%
berlin52	7542	7542.00	0.00
brazil58	25395	25395.00	0.00
st70	731	675.00	7.66
eil76	543	538.00	0.92
gr96	57929	55209.00	4.70
rat99	1256	1211.00	3.58
kroD100	21765	21294.00	2.16
kroE100	24703	22068.00	10.67
kroA100	21767	21282.00	2.23
kroC100	22011	20749.00	5.73
kroB100	23106	22141.00	4.18
rd100	8134	7910.00	2.75
eil101	629	629.00	0.00
lin105	14504	14379.00	0.86
pr107	44303	44303.00	0.00
gr120	7145	6942.00	2.84
pr124	59824	59030.00	1.33
bier127	126457	118282.00	6.46
ch130	6276	6110.00	2.64
pr136	98632	96772.00	1.89
gr137	72341	69853.00	3.44
pr144	59282	58537.00	1.26
ch150	6613	6528.00	1.29
kroA150	28090	26524.00	5.57
kroB150	27595	26130.00	5.31
pr152	75799	73682.00	2.79
brg180	1950	1950.00	0.00
rat195	2479	2323.00	6.29
d198	16001	15780.00	1.38
kroB200	32165	29437.00	8.48
kroA200	30516	29368.00	3.76
gr202	40326	40160.00	0.41
ts225	146110	126643.00	13.32
tsp225	4301	3916.00	8.95
pr226	81845	80369.00	1.80
gr229	140484	134602.00	4.19
gil262	2460	2378.00	3.33
pr264	49670	49135.00	1.08
a280	2784	2579.00	7.36

continued on next page

continued from previous page

Instance	Compatible	TSP Optimum	GAP%
pr299	50871	48191.00	5.27
lin318	42736	42029.00	1.65
rd400	15704	15281.00	2.69
fl417	12112	11861.00	2.07
gr431	178367	171414.00	3.90
pr439	109320	107217.00	1.92
pcb442	52166	50778.00	2.66
d493	36769	35002.00	4.81
ali535	208312	202339.00	2.87
u574	39106	36905.00	5.63
rat575	7136	6773.00	5.09
d657	50705	48912.00	3.54
gr666	302923	294358.00	2.83
u724	43944	41910.00	4.63
rat783	9307	8806.00	5.38
pcb1173	60467	56892.00	5.91

Table 7.1: Gap for problems in TSPLIB

We can observe that the largest gap was 13.32% for `ts225` and the smallest gap was 0, which means that the heuristic is capable of finding the optimal solution.

The average gap was 3.08%, which is quite a good performance for a tour construction heuristic.

As an example, we gathered some comparison results in table 7.2 from the DIMACS challenge [4] pages for tour construction heuristics and problem `pcb1173`. The compatible tour heuristic performed better than all but one of the tour construction heuristic listed on the results page for this particular problem.

Algorithm	GAP%
Greedy	18.15
CONCORDE Greedy	19.76
Bentley "multi-fragment"	18.29
Boruvka (CONCORDE)	17.47
"Quick"-Boruvka (CONCORDE)	14.82
Nearest neighbor (CONCORDE)	28.26
Farthest Insertion	15.13
Farthest Addition	42.47
Farthest Augmented Addition	15.27
Clarke-Wright Savings	11.45
Golden-Stewart CCA	11.91

continued on next page

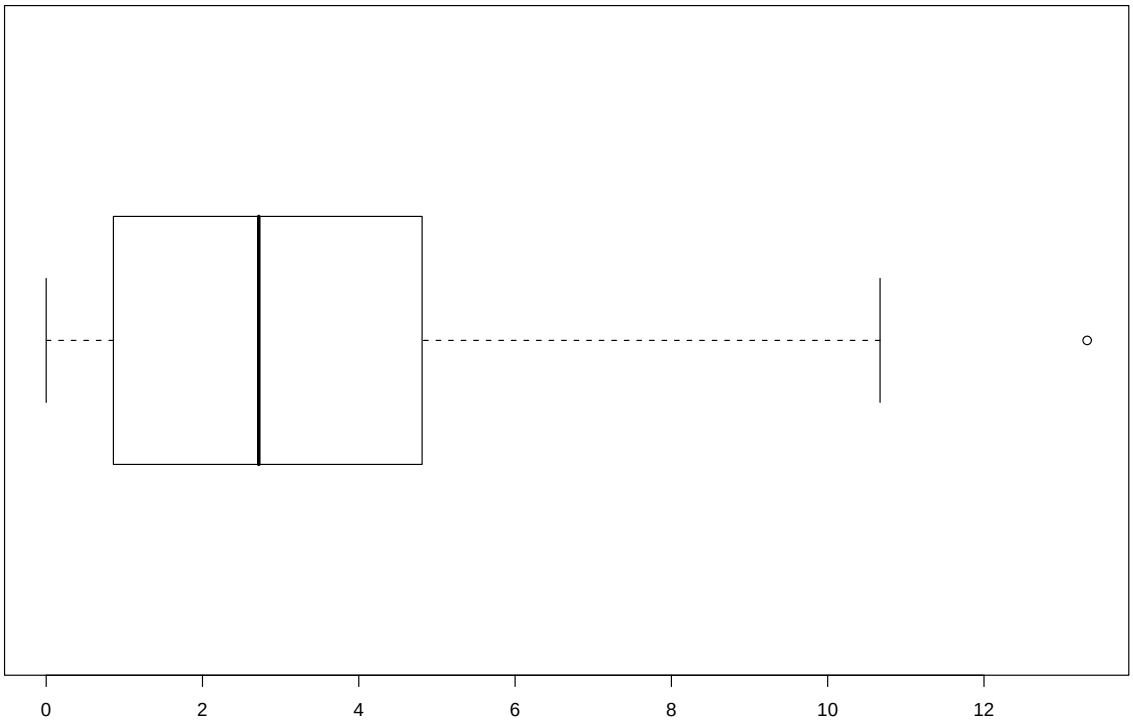

Figure 7.1: Box plot of gaps in table 7.1

continued from previous page	
Algorithm	GAP%
Christo-S	12.24
Christo-G	8.20
HK one-tree	**5.29**
MST approx, greedy	7.84
MST, half-LK	12.78
Compatible Tour	5.91

Table 7.2: DIMACS Gaps for tour construction heuristics on pcb1173

In figure 7.1 we see that the population is well represented by the average and that the largest gap we found is an outlier. Most of the values are in fact in the range from 1 to 5.

7.2 Times for the B&C and price algorithm

In table 7.3 we gathered some results obtained by running our complete compatible-tour B&C code with pricing on a PIII-M 1133MHz laptop, using CPLEX 10.0 and

Linux as the underlying Operating System. The software was coded in ansi-C programming language.

Times are in seconds.

The columns are:

- **Instance** the TSPLIB instance
- T_{tot} the total time to find the compatible tour
- T_{x^*} the time to get the extremum point of the SEP x^*
- T_{cactus} the time to generate the cactus
- $T_{mincuts}$ the time to insert the minimum cuts in the LP
- T_{cut} the time spent for separating violated cuts using CONCORDE callable library
- T_{check} the time spent for proving the best integral solution
- T_{price} the time spent for pricing
- $N_{branches}$ the total number of branching nodes explored

Instance	T_{tot}	T_{x^*}	T_{cactus}	$T_{mincuts}$	T_{cut}	T_{check}	T_{price}	N_{branch}
ulysses16	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0
gr17	0.04	0.03	0.00	0.00	0.01	0.00	0.00	0
gr21	0.03	0.02	0.00	0.00	0.00	0.00	0.00	0
ulysses22	0.06	0.04	0.00	0.00	0.02	0.00	0.00	0
gr24	0.04	0.02	0.00	0.01	0.01	0.00	0.00	0
fri26	0.08	0.05	0.00	0.00	0.02	0.00	0.00	0
bays29	0.04	0.01	0.00	0.00	0.01	0.00	0.00	0
bayg29	0.04	0.02	0.00	0.00	0.01	0.00	0.00	0
dantzig42	0.11	0.04	0.00	0.01	0.00	0.00	0.01	0
swiss42	0.10	0.03	0.00	0.01	0.01	0.00	0.00	0
att48	0.20	0.08	0.00	0.01	0.07	0.00	0.02	2
hk48	0.19	0.10	0.00	0.02	0.04	0.00	0.01	0
gr48	0.13	0.06	0.00	0.01	0.03	0.00	0.01	0
eil51	0.12	0.06	0.00	0.01	0.02	0.00	0.00	0
berlin52	0.16	0.04	0.00	0.00	0.01	0.00	0.00	0
brazil58	0.26	0.10	0.00	0.01	0.02	0.00	0.03	0
st70	0.24	0.12	0.00	0.01	0.05	0.00	0.04	0
eil76	0.26	0.08	0.00	0.02	0.01	0.00	0.01	0
pr76	0.26	0.08	0.00	0.01	0.06	0.00	0.02	4

continued on next page

continued from previous page

Instance	T_{tot}	T_{x^*}	T_{cactus}	$T_{mincuts}$	T_{cut}	T_{check}	T_{price}	N_{branch}
gr96	0.62	0.23	0.00	0.02	0.06	0.00	0.07	0
rat99	0.51	0.08	0.00	0.04	0.06	0.00	0.02	0
kroC100	0.59	0.31	0.00	0.02	0.05	0.00	0.12	0
kroA100	0.41	0.15	0.00	0.03	0.04	0.00	0.04	0
kroD100	0.63	0.37	0.00	0.02	0.08	0.00	0.14	0
rd100	0.78	0.42	0.00	0.04	0.11	0.00	0.19	0
kroE100	0.55	0.19	0.00	0.02	0.05	0.00	0.06	0
kroB100	0.62	0.32	0.00	0.02	0.04	0.00	0.15	0
eil101	1.01	0.19	0.00	0.02	0.06	0.00	0.08	2
lin105	1.05	0.26	0.00	0.02	0.04	0.00	0.09	0
pr107	1.59	0.46	0.00	0.02	0.05	0.00	0.24	0
gr120	1.00	0.47	0.00	0.08	0.12	0.00	0.25	0
pr124	1.00	0.46	0.00	0.04	0.08	0.00	0.22	0
bier127	1.38	0.54	0.00	0.04	0.10	0.00	0.26	0
ch130	1.22	0.56	0.00	0.05	0.24	0.00	0.22	0
pr136	3.36	0.72	0.01	0.18	1.76	1.18	0.23	18
gr137	1.29	0.42	0.01	0.05	0.09	0.00	0.15	0
pr144	1.86	1.19	0.00	0.04	0.20	0.00	0.52	0
ch150	1.66	1.00	0.00	0.10	0.14	0.00	0.52	0
kroA150	1.58	0.78	0.00	0.06	0.13	0.00	0.38	0
kroB150	1.26	0.49	0.00	0.06	0.10	0.00	0.26	0
pr152	4.34	3.41	0.01	0.08	0.45	0.00	1.50	0
u159	2.68	0.56	0.00	0.04	0.08	0.00	0.32	0
brg180	3.22	0.83	0.02	0.33	0.73	0.50	0.12	28
rat195	5.04	0.70	0.01	0.17	1.90	0.31	0.37	82
d198	5.12	3.25	0.01	0.12	0.48	0.00	1.68	4
kroA200	2.87	1.37	0.01	0.12	0.23	0.00	0.87	0
kroB200	2.06	0.76	0.01	0.09	0.12	0.00	0.40	0
gr202	7.63	1.28	0.00	0.08	0.11	0.00	0.83	0
tsp225	3.91	1.50	0.01	0.12	0.26	0.00	0.96	2
ts225	1.70	0.07	0.00	0.08	0.02	0.00	0.01	0
pr226	25.62	19.25	0.01	0.14	0.82	0.00	13.42	0
gr229	6.02	2.00	0.01	0.28	2.10	0.00	1.10	6
gil262	9.26	3.74	0.01	0.12	0.26	0.00	2.64	0
pr264	21.53	2.47	0.02	0.14	0.19	0.00	1.66	0
a280	5.82	1.11	0.01	0.14	0.26	0.06	0.74	6
pr299	10.71	3.00	0.01	0.13	0.22	0.00	2.33	0
lin318	26.30	17.12	0.02	0.22	1.14	0.00	11.58	0
rd400	35.19	10.04	0.02	0.36	1.29	0.00	7.40	4

continued on next page

continued from previous page

Instance	T_{tot}	T_{x^*}	T_{cactus}	$T_{mincuts}$	T_{cut}	T_{check}	T_{price}	N_{branch}
fl417	83.31	26.15	0.02	0.36	0.87	0.00	21.33	4
gr431	31.99	11.01	0.03	0.82	1.94	0.00	8.30	0
pr439	49.94	17.69	0.03	0.83	1.98	0.00	13.50	0
pcb442	25.09	3.22	0.02	0.45	0.81	0.06	2.34	14
d493	43.78	19.39	0.02	0.58	1.00	0.00	15.96	2
ali535	74.42	56.56	0.03	0.92	1.37	0.00	49.10	0
u574	62.94	32.68	0.02	0.83	1.34	0.00	28.33	0
rat575	24.57	4.95	0.03	0.87	0.95	0.00	3.58	0
d657	65.42	27.68	0.04	1.42	1.90	0.00	23.46	0
gr666	140.24	28.08	0.05	3.47	60.26	5.39	23.57	32
u724	99.98	50.12	0.06	1.51	1.85	0.02	44.01	4
rat783	133.73	23.09	0.06	1.38	0.90	0.00	19.71	0
pcb1173	336.15	70.92	0.08	4.15	1.57	0.00	63.44	0
nrw1379	722.05	143.85	0.18	12.43	231.07	8.91	131.84	24
d1655	3976.12	1615.66	0.20	10.55	7.95	0.00	1555.01	0
	Totals	**2214.13**	**1.08**	**44.43**	**330.42**	**16.43**	**2055.77**	

Table 7.3: Times for Branch & Cut

It is clear from Figure 7.2 that most of the time is spent finding the initial fractional solution x^* and for pricing, while the time spent in implementing the core of the heuristic is comparatively very small.

In our experience the time for pricing was mainly spent when proving that there were no negative reduced costs variables, which requires to test every variable not in the *tableau*.

We didn't include in the figure the time to build the cactus tree representation of the minimum cuts because it was too small, compared to the others.

From figure 7.3 we can see that the time to solve the problem grows exponentially with the size of the problem, as we expected.

The number of branches was very low in general, with a maximum of 82 branches to optimality for `rat195`. Most of the problems found the optimal solution at the root node after having inserted the compatibility constraints and cutting.

7.3 Dynamic Programming algorithm

In this section we present the performance of the Dynamic programming algorithm.

The columns are:

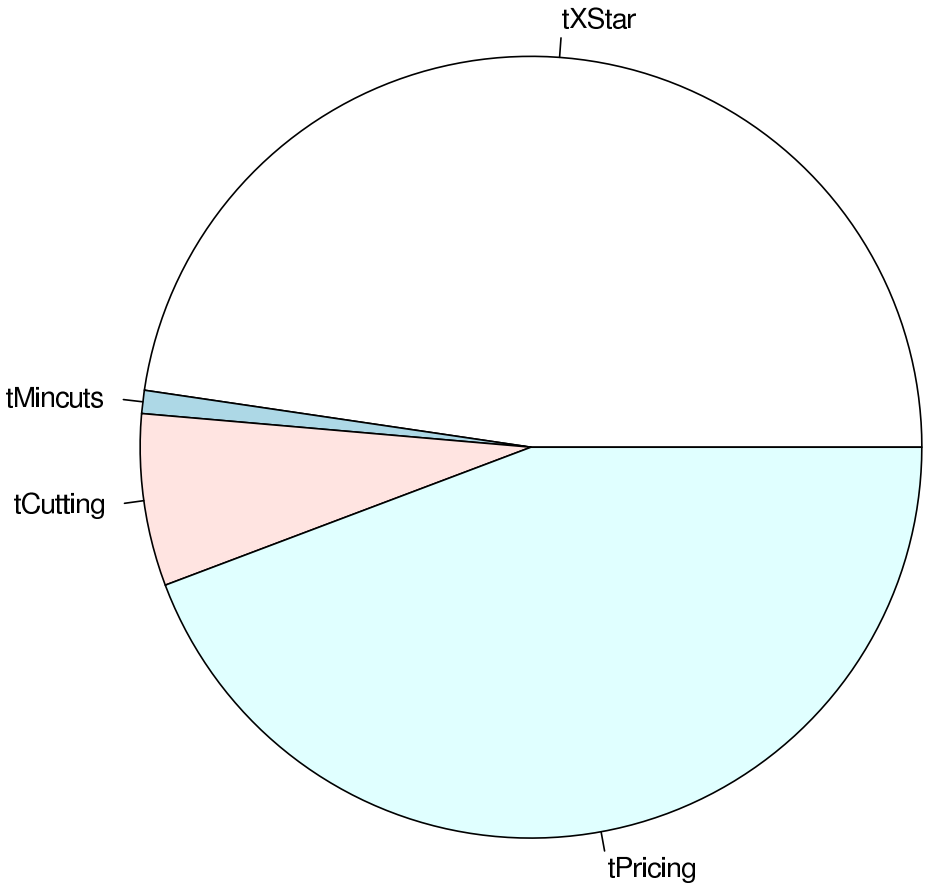

Figure 7.2: Times for steps in table 7.3

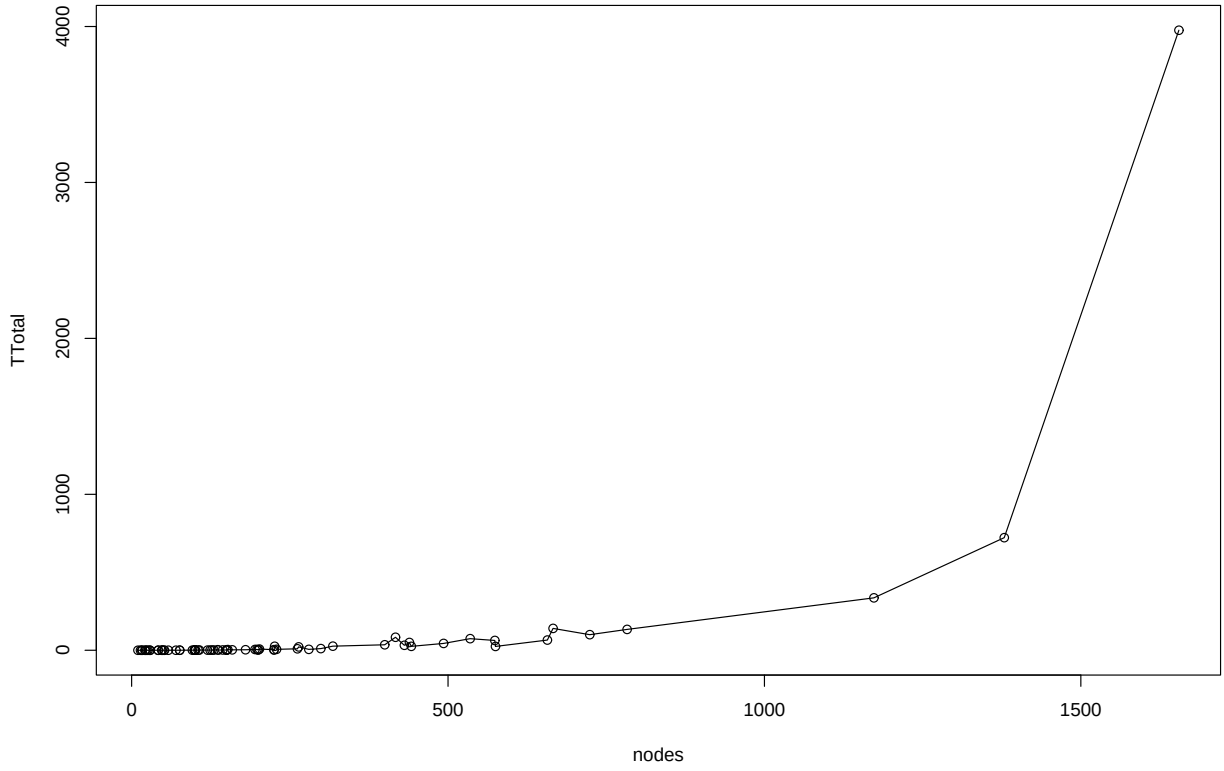

Figure 7.3: T_{tot} versus number of nodes in table 7.3

- **Instance** the instance from TSPLIB
- **Onechains** the number of chains of one that have been shrunk
- **maxDeg** the maximum degree for a node in the cactus
- $N_{subproblems}$ the number of generated subproblems
- T_{tot} the total time spent

The time figure cant't be directly compared to the one obtained by B&C because the DP algorithm was coded in the Python programming language for testing purposes. We can expect at least a $1/10$ ratio over the performance of optimized C code. Note that the DP code started from the cactus representation of minimum cuts, so it doesn't take into account the time for finding x^*.

Instance	Onechains	maxDeg	$N_{subproblems}$	T_{tot}
gr10	5	4	82	0.521472
att15	4	4	34	0.294400
gr24	5	4	82	0.530799
bays29	3	4	16	0.195719
bayg29	4	4	30	0.291470
dantzig42	8	4	798	2.940290
swiss42	5	4	56	0.570392
hk48	8	4	226	1.127665
att48	9	7	962	2.869591
gr48	9	4	678	2.095302
eil51	5	4	62	0.655221
brazil58	3	4	16	0.230893
st70	7	4	66	0.639436
pr76	9	12	2656	6.331394
eil76	4	4	34	0.462664
gr96	7	4	162	0.990478
rat99	15	4	1654	10.470588
rd100	15	4	3220	14.872333
kroC100	9	4	336	2.337255
kroA100	13	4	862	3.913415
kroE100	3	4	16	0.341285
kroD100	12	4	782	4.005255
kroB100	9	4	336	1.627935
eil101	11	8	4284	15.830658
lin105	4	4	34	0.588565
gr120	24	8	76330	1756.254541

continued on next page

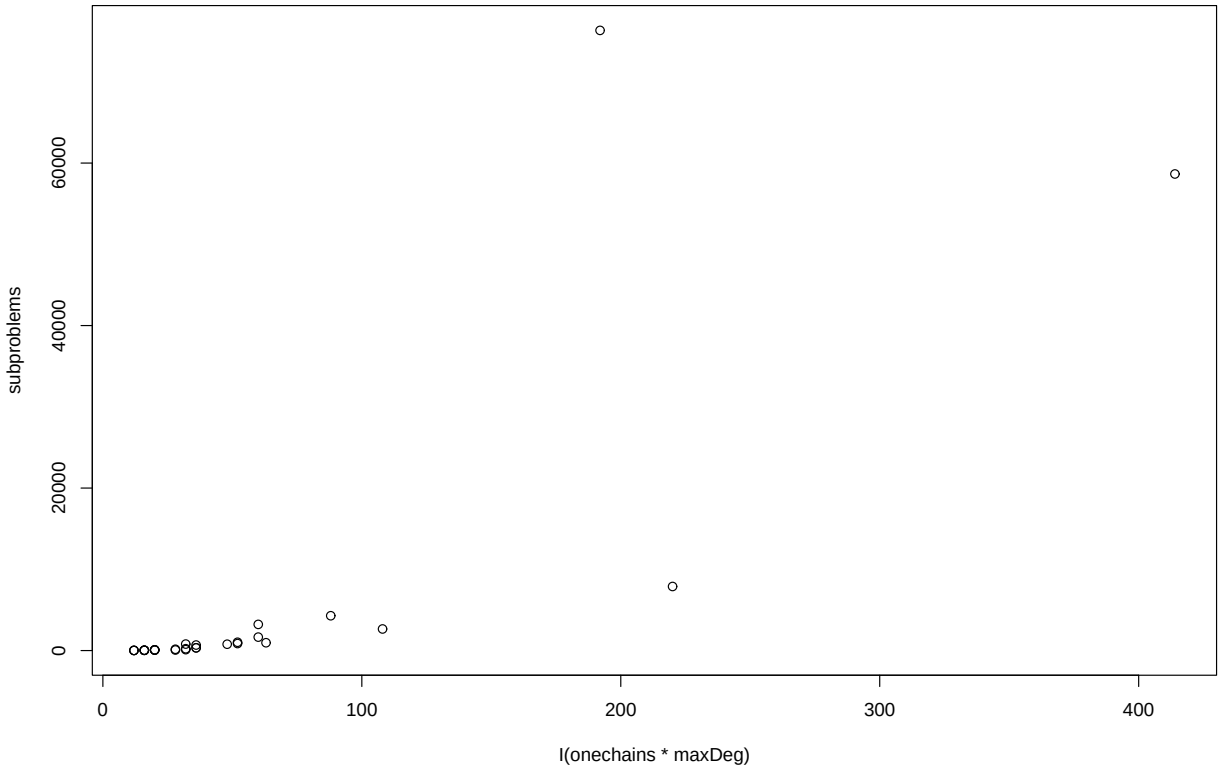

Figure 7.4: Subproblems versus onechains×maxDeg for table 7.4

continued from previous page				
Instance	Onechains	maxDeg	$N_{subproblems}$	T_{tot}
pr124	8	4	112	2.096782
bier127	13	4	1022	6.247325
ch130	20	11	7886	66.639272
pr136	46	9	58656	5852.500642

Table 7.4: Dynamic Programming algorithm

We can see that the time increases linearly with the number of subproblems that are generated. The number of subproblems is directly affected by the number of one-chains and the maximum degree for a node as one can see in figure 7.4

Chapter 8

Conclusions

In this work we presented a novel heuristic for the TSP, called the Compatible Tour heuristic. The algorithm is a tour construction one, and it is strongly based on a fundamental LP relaxation of the TSP, which constrains define the Subtour Elimination Polytope.

We studied the problem from a theoretical point of view and found several properties that reduce the steps required for the implementation of the algorithm.

We proved the $\mathcal{NP}$-hard computational complexity of the heuristic and pinpointed a class of "bad" instances which have a relative gap that goes up to $5/3$.

We implemented three different algorithmic approaches to the solution of the problem, the first based on the lagrangean relaxation of the constraints, the second on a complete Branch&Cut with pricing solution and the third using Dynamic Programming.

We tested the software against a wide range of benchmark problems from TSPLIB, finding that it performed quite well on a large number of instances, having a small average and absolute gap compared to other known tour-construction heuristics.

Bibliography

[1] K.M. Wenger (2002) A New Approach to Cactus Construction Applied to TSP Support Graphs. *9th International IPCO Conference Proc.*, LNCS 2337, Springer, 109–126

[2] D. Applegate, R.E. Bixby, V. Chvátal & W. Cook (2003) CONCORDE TSP Solver `http://www.tsp.gatech.edu/concorde.html`

[3] G. Reinelt, TSPLIB `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

[4] D.S. Johnson, DIMACS Challenge `http://www.research.att.com/~dsj/chtsp/index.html`

[5] K.S. Booth & G.S. Lueker (1976) Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comp. Sys. Sci.*, 12, 335–379.

[6] S. Boyd & W. Pulleyblank (1991) Optimizing over the subtour polytope of the travelling salesman problem. *Math. Program.*, 49, 163–187.

[7] R.E. Burkhard, V.G. Deineko & G.J. Woeginger (1998) The travelling salesman and the PQ-tree. *Math. Oper. Res.*, 23, 613–623.

[8] N. Christofides (1976) Worst-case analysis of a new heuristic for the travelling salesman problem. *Report 388*, Graduate School of Industrial Administration, Carnegie Mellon University.

[9] V. Deineko & A. Tiskin (2006) Double-tree approximations for metric TSP: is the best one good enough? *Technical Report*.

[10] E.A. Dinitz, A.V. Karzanov & M.V. Lomonosov (1976) A structure for the system of all minimum cuts of a graph. In A.A. Fridman (ed.) *Studies in Discrete Optimization*, Nauka, Moscow, pp. 290–306 (in Russian).

[11] L. Fleischer (1999) Building chain and cactus representations of all minimum cuts from Hao-Orlin in the same asymptotic run time. *J. of Algorithms*, 33, 51–72.

[12] M.R. Garey, D.S. Johnson & L. Stockmeyer (1974) Some simplified $\mathcal{NP}$-complete problems. In *Proc. 6th annual ACM Symposium on Theory of Computing*, pp. 47–63.

[13] G. Gutin & A.P. Punnen (eds) (2002) *The Traveling Salesman Problem and its Variations*. Kluwer.

[14] M. Held & R.M. Karp (1970) The traveling salesman problem and minimum cost spanning trees. *Oper. Res.*, 18, 1138–1162.

[15] E. Lawler, J. Lenstra, A. Rinnooy Kan & D. Shmoys (eds.), *The Traveling Salesman Problem*. John Wiley & Sons, Chichester.

[16] D.J. Rosenkrantz, R.E. Stearns & P.M. Lewis (1977) An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6, 563-581.

[17] D.L. Applegate, R.E. Bixby, V. Chvátal & W.J. Cook (1995) *Finding cuts in the TSP (a preliminary report)*. Technical Report 95–05, DIMACS, Rutgers University, New Brunswick, NJ.

[18] D.L. Applegate, R.E. Bixby, V. Chvátal & W.J. Cook (1998) On the solution of traveling salesman problems. *Documenta Mathematica* Extra Volume ICM III 645–656.

[19] D.L. Applegate, R.E. Bixby, V. Chvátal & W.J. Cook (2001) TSP cuts which do not conform to the template paradigm. In M. Jünger and D. Naddef (eds.) *Computational Combinatorial Optimization*. Springer.

[20] D.L. Applegate, R.E. Bixby, V. Chvátal & W.J. Cook (2003) Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. *Math. Program.*, 97, 91–153.

[21] D.L. Applegate, R.E. Bixby, V. Chvátal & W.J. Cook (2006) *The Traveling Salesman Problem: A Computational Study.* Princeton University Press.

[22] E. Balas (1989) The asymmetric assignment problem and some new facets of the traveling salesman polytope. *SIAM J. Discr. Math.*, 2, 425–451.

[23] E. Balas, R. Carr, M. Fischetti & N. Simonetti (2006) New facets of the STS polytope generated from known facets of the ATS polytope. *Discr. Opt.*, 3, 3-19.

[24] E. Balas & M. Fischetti (1992) The fixed out-degree 1-arborescence polytope. *Math. Oper. Res.*, 17, 1001–1018.

[25] E. Balas & M. Fischetti (1993) A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Math. Program.*, 58, 325–352.

[26] E. Balas & M. Fischetti (1997) On the monotonization of polyhedra. *Math. Program.*, 78, 59–84.

[27] E. Balas & M. Fischetti (1999) Lifted cycle inequalities for the asymmetric traveling salesman problem. *Math. Oper. Res.*, 24, 273–292.

[28] E. Balas & M. Fischetti (2002) Polyhedral theory for the asymmetric traveling salesman problem. In G. Gutin & A.P. Punnen (eds.) *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers.

[29] S.C. Boyd, S. Cockburn & D. Vela (2006) On the domino-parity inequalities for the TSP. *Math. Program.*, to appear.

[30] S.C. Boyd & W.H. Cunningham (1991) Small traveling salesman polytopes. *Math. Oper. Res.*, 16, 259–271.

[31] S.C. Boyd, W.H. Cunningham, M. Queyranne & Y. Wang (1995) Ladders for travelling salesmen. *SIAM J. Optim.*, 5, 408–420.

[32] S. Boyd & G. Labonté (2002) Finding the exact integrality gap for small travelling salesman problems. In W.J. Cook & A.S. Schulz (eds.) *Integer Programming and Combinatorial Optimization 9*, Lecture Notes in Computer Science 2337. Berlin: Springer-Verlag.

[33] S.C. Boyd & W.R. Pulleyblank (1990) Optimizing over the subtour polytope of the traveling salesman problem. *Math. Program.*, 49, 163–187.

[34] A. Caprara & M. Fischetti (1996) $\{0, 1/2\}$-Chvátal-Gomory cuts. *Math. Program.*, 74, 221–235.

[35] A. Caprara, M. Fischetti & A.N. Letchford (2000) On the separation of maximally violated mod-k cuts. *Math. Program.*, 87, 37–56.

[36] A. Caprara & A.N. Letchford (2003) On the separation of split cuts and related inequalities. *Math. Program.*, 94, 279–294.

[37] R.D. Carr (1996) Separating over classes of TSP inequalities defined by 0 node-lifting in polynomial time. In W.H. Cunningham, S.T. McCormick & M. Queyranne (eds.) *Integer Programming and Combinatorial Optimization 5*, Lecture Notes in Computer Science 1084. Berlin: Springer-Verlag.

[38] R.D. Carr (1997) Separating clique tree and bipartition inequalities having a fixed number of handles and teeth in polynomial time. *Math. Oper. Res.*, 22, 257–265.

[39] R.D. Carr (2000) Some results on node lifting of TSP inequalities. *J. Comb. Opt.*, 4, 395–414.

[40] R.D. Carr (2004) Separation algorithms for classes of STSP inequalities arising from a new STSP relaxation. *Math. Oper. Res.*, 29, 80–91.

[41] S. Chopra & G. Rinaldi (1990) The graphical asymmetric traveling salesman polyhedron. In R. Kannan & W.R. Pulleyblank (eds.) *Integer Programming and Combinatorial Optimization 1*, University of Waterloo Press.

[42] S. Chopra & G. Rinaldi (1996) The graphical asymmetric traveling salesman polyhedron: symmetric inequalities. *SIAM J. Discr. Math.*, 9, 602–624.

[43] T. Christof, M. Jünger, G. Reinelt (1991) A complete description of the traveling salesman polytope on 8 nodes. *Oper. Res. Lett.*, 10, 497–500.

[44] T. Christof & G. Reinelt (1995) Parallel cutting plane generation for the TSP. In P. Fritzson & L. Finno (eds.) *Parallel Programming and Applications*. Amsterdam: IOS Press, 163–169.

[45] T. Christof & G. Reinelt (1996) Combinatorial optimization and small polytopes. *Top (J. Span. Stat. & O.R. Soc.)*, 4, 1–64.

[46] V. Chvátal (1973) Edmonds polytopes and weakly Hamiltonian graphs. *Math. Program.*, 5, 29–40.

[47] V. Chvátal, W. Cook & M. Hartmann (1989) On cutting plane proofs in combinatorial optimization. *Lin. Alg. Appl.*, 114/115, 455–499.

[48] J.-M. Clochard & D. Naddef (1993) Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem. In G. Rinaldi & L.A. Wolsey (eds.) *Integer Programming and Combinatorial Optimization 3*, CORE, Catholic University of Louvain, 291–311.

[49] W. Cook, D. Espinoza & M. Goycoolea (2005) A study of domino-parity and k-parity constraints for the TSP. In M. Junger & V. Kaibel (eds.) *Integer Programming and Combinatorial Optimization 11*. Lecture Notes in Computer Science vol. 3509. Berlin: Springer-Verlag.

[50] W. Cook, D. Espinoza & M. Goycoolea (2006) Computing with domino-parity inequalities for the TSP. *INFORMS J. Computing*, to appear.

[51] W. Cook, D. Espinoza & M. Goycoolea (2007) A generalization of domino-parity constraints to multiple-handle configurations. *Working paper*.

[52] G. Cornuéjols, J. Fonlupt & D. Naddef (1985) The travelling salesman on a graph and some related integer polyhedra. *Math. Program.*, 33, 1–27.

[53] G. Cornuéjols, D. Naddef & W.R. Pulleyblank (1985) The traveling salesman problem in graphs with 3-edge cutsets. *J. of the ACM*, 32, 383–410.

[54] H. Crowder & M.W. Padberg (1980) Solving large-scale symmetric traveling salesman problems to optimality. *Mgt. Sci.*, 26, 495–509.

[55] W.H. Cunningham & Y. Wang (2000) Restricted 2-factor polytopes. *Math. Program.*, 87, 87–111.

[56] G.B. Dantzig, D.R. Fulkerson & S.M. Johnson (1954) Solution of a large-scale traveling salesman problem. *Oper. Res.*, 2, 393–410.

[57] R. Euler & H. Le Verge (1995) Complete linear descriptions of small asymmetric traveling salesman polytopes. *Discr. Appl. Math.*, 62, 193–208.

[58] M. Fischetti (1991) Facets of the asymmetric traveling salesman polytope. *Math. Oper. Res.*, 16, 42–56.

[59] M. Fischetti (1992) Three lifting theorems for the asymmetric traveling salesman polytope. E. Balas, G. Cornuéjols, R. Kannan (eds.) *Integer Programming and Combinatorial Optimization 2*, Pittsburgh. Carnegie Mellon University, 260–273.

[60] M. Fischetti (1995) Clique tree inequalities define facets of the asymmetric traveling salesman polytope. *Discr. Appl. Math.*, 56, 9–18.

[61] M. Fischetti, A. Lodi & P. Toth (2002) Exact methods for the asymmetric traveling salesman problem. In G. Gutin & A. Punnen (eds.) *The Traveling Salesman Problem and its Variations*. Kluwer Academic Publishers.

[62] M. Fischetti, A. Lodi & P. Toth (2003) Solving real-world ATSP instances by branch-and-cut. In M. Jünger, G. Reinelt & G. Rinaldi (eds.) *Combinatorial Optimization - Eureka, You Shrink!* Lecture Notes In Computer Science vol. 2570. Berlin: Springer-Verlag.

[63] M. Fischetti & P. Toth (1997) A polyhedral approach to the asymmetric traveling salesman problem. *Mgt. Sci.*, 43, 1520–1536.

[64] L.K. Fleischer, A.N. Letchford & A. Lodi (2006) Polynomial-time separation of a superclass of simple comb inequalities. *Math. Oper. Res.*, 31, 696–713.

[65] L. Fleischer & É. Tardos (1999) Separating maximally violated comb inequalities in planar graphs. *Math. Oper. Res.*, 24, 130–148.

[66] B. Fleischmann (1985) A cutting plane procedure for the travelling salesman problem on a road network. *Eur. J. Opl Res.*, 21, 307–317.

[67] B. Fleischmann (1987) *Cutting planes for the symmetric travelling salesman problem*. Technical report, Universität Hamburg.

[68] B. Fleischmann (1988) A new class of cutting planes for the symmetric travelling salesman problem. *Math. Program.*, 40, 225–246.

[69] J. Fonlupt & A. Nachef (1993) Dynamic programming and the graphical traveling salesman problem. *J. Assoc. Comput. Mach.*, 40, 1165–1187.

[70] J. Fonlupt & D. Naddef (1992) The traveling salesman problem in graphs with excluded minors. *Math. Program.*, 53, 147–172.

[71] M.X. Goemans (1995) Worst-case comparison of valid inequalities for the TSP. *Math. Program.*, 69, 335–349.

[72] M. Grötschel (1980) On the monotone symmetric travelling salesman problem: hypohamiltonian / hypotracable graphs and facets. *Math. Oper. Res.*, 5, 285–292.

[73] M. Grötschel (1980) On the symmetric travelling salesman problem: solution of a 120–city problem. *Math. Program. Study*, 12, 61–77.

[74] M. Grötschel & O. Holland (1991) Solution of large-scale symmetric traveling salesman problems. *Math. Program.*, 51, 141–202.

[75] M. Grötschel, L. Lovász & A.J. Schrijver (1988) *Geometric Algorithms and Combinatorial Optimization*. Berlin: Springer-Verlag.

[76] M. Grötschel & M.W. Padberg (1975) Partial linear characterizations of the asymmetric travelling salesman polytope. *Math. Program.*, 8, 378–381.

[77] M. Grötschel & M.W. Padberg (1979) On the symmetric travelling salesman problem I: inequalities. *Math. Program.*, 16, 265–280.

[78] M. Grötschel & M.W. Padberg (1979) On the symmetric travelling salesman problem II: lifting theorems and facets. *Math. Program.*, 16, 281–302.

[79] M. Grötschel & M.W. Padberg (1985) Polyhedral theory. In E. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shmoys (eds.). *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, 251–305.

[80] M. Grötschel & W.R. Pulleyblank (1986) Clique tree inequalities and the symmetric traveling salesman problem. *Math. Oper. Res.*, 11, 537–569.

[81] M. Grötschel & Y. Wakabayashi (1981) On the structure of the monotone asymmetric travelling salesman polytope I: hypohamiltonian facets. *Discr. Math.*, 34, 43–59.

[82] M. Grötschel & Y. Wakabayashi (1981) On the structure of the monotone asymmetric travelling salesman polytope II: hypotraceable facets. *Math. Program. Study*, 14, 77–97.

[83] M. Hartmann, M. Queyranne & Y. Wang (1999) On the Chvátal rank of certain inequalities. In G. Cornuéjols, R.E. Burkard & G.J. Woeginger (eds.) *Integer Programming and Combinatorial Optimization 7*, Lecture Notes in Computer Science 1610. Berlin: Springer-Verlag.

[84] M. Jünger, G. Reinelt, G. Rinaldi (1995) The traveling salesman problem. In M. Ball, T. Magnanti, C. Monma & G. Nemhauser (eds.). *Network Models*, Handbooks in Operations Research and Management Science, 7, Elsevier Publisher B.V., Amsterdam, 225–330.

[85] M. Jünger, G. Reinelt & G. Rinaldi (1997) The traveling salesman problem. In M. Dell'Amico, F. Maffioli & S. Martello (eds.) *Annotated Bibliographies in Combinatorial Optimization*. Chichester: Wiley.

[86] M. Jünger, G. Reinelt & S. Thienel (1994) Provably good solutions for the traveling salesman problem. *Z. Oper. Res.*, 40, 183–217.

[87] R.M. Karp & C.H. Papadimitriou (1982) On linear characterizations of combinatorial optimization problems. *SIAM J. Comput.*, 11, 620–632.

[88] A.N. Letchford (2000) Separating a superclass of comb inequalities in planar graphs. *Math. Oper. Res.*, 25, 443–454.

[89] A.N. Letchford (2001) On disjunctive cuts for combinatorial optimization. *J. Comb. Opt.*, 5, 299–315.

[90] A.N. Letchford (2003) Binary clutter inequalities for integer programs. *Math. Program.*, 98, 201–221.

[91] A.N. Letchford & A. Lodi (2002) Polynomial-time separation of simple comb inequalities. In W.J. Cook & A.S. Schulz (eds.), *Integer Programming and Combinatorial Optimization 9*. Lecture Notes in Computer Science vol 2337. Berlin: Springer-Verlag.

[92] A.N. Letchford & N.A. Pearson (2006) Exploiting planarity in separation routines for the symmetric traveling salesman problem. *Discr. Opt.*, to appear.

[93] A.N. Letchford, G. Reinelt & D.O. Theis (2004) A faster exact separation algorithm for blossom inequalities. In G. Nemhauser & D. Bienstock (eds.) *Integer Programming and Combinatorial Optimization 10*. Lecture Notes in Computer Science vol. 3064. Berlin: Springer-Verlag.

[94] P. Miliotis (1976) Integer programming approaches to the travelling salesman problem. *Math. Program.*, 10, 367–378.

[95] P. Miliotis (1978) Using cutting planes to solve the symmetric travelling salesman problem. *Math. Program.*, 15, 177–188.

[96] J.-F. Maurras & V.H. Nguyen (2003) A procedure of facet composition for the symmetric traveling salesman polytope. In M. Jünger, G. Reinelt & G. Rinaldi (eds.) *Combinatorial Optimization - Eureka, You Shrink!* Lecture Notes In Computer Science vol. 2570. Berlin: Springer-Verlag.

[97] D. Naddef (1990) Handles and teeth in the symmetric traveling salesman polytope. In W. Cook & P.D. Seymour (eds.) *Polyhedral Combinatorics*. Baltimore: American Mathematical Society.

[98] D. Naddef (1992) The binested inequalities for the symmetric traveling salesman polytope. *Math. Oper. Res.*, 17, 882–900.

[99] D. Naddef (2002) Polyhedral theory and branch-and-cut algorithms for the TSP. In G. Gutin & A.P. Punnen (eds), *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers.

[100] D. Naddef (2004) The domino inequalities for the symmetric traveling salesman problem. In M. Grötschel (ed.) *The Sharpest Cut: the Impact of Manfred Padberg and His Work*. MPS-SIAM Series on Optimization vol. 4.

[101] D. Naddef & J.-M. Clochard (1994) *Some fast and efficient heuristics for comb separation in the symmetric traveling salesman problem*. Technical Report RR–941, ARTEMIS, Grenoble.

[102] D. Naddef & Y. Pochet (2001) The symmetric traveling salesman polytope revisited. *Math. Oper. Res.*, 26, 700-722.

[103] D. Naddef & G. Rinaldi (1988) *The symmetric traveling salesman polytope: New facets from the graphical relaxation*. Technical Report 248, IASI-CNR, Rome.

[104] D. Naddef & G. Rinaldi (1991) The symmetric traveling salesman polytope and its graphical relaxation: composition of valid inequalities. *Math. Program.*, 51, 359–400.

[105] D. Naddef & G. Rinaldi (1992) The crown inequalities for the symmetric traveling salesman polytope. *Math. Oper. Res.*, 17, 308–326.

[106] D. Naddef & G. Rinaldi (1993) The graphical relaxation: a new framework for the symmetric travelling salesman polytope. *Math. Program.*, 58, 53–88.

[107] D. Naddef & S. Thienel (2002) Efficient separation routines for the symmetric traveling salesman problem I: general tools and comb separation. *Math. Program.*, 92, 237–255.

[108] D. Naddef & S. Thienel (2002) Efficient separation routines for the symmetric traveling salesman problem II: separating multi-handle inequalities. *Math. Program.*, 92, 257–283.

[109] D. Naddef & E. Wild (2003) The domino inequalities: facets for the symmetric traveling salesman polytope. *Math. Program.*, 98, 223–251.

[110] H. Nagamochi, T. Ono & T. Ibaraki (1994) Implementing an efficient minimum cut algorithm. *Math. Program.*, 67, 325-341.

[111] G.L. Nemhauser & L.A. Wolsey (1988) *Integer and Combinatorial Optimization*. New York: Wiley.

[112] M. Oswald, G. Reinelt & D.O. Theis (2005) Not every GTSP facet induces an STSP facet. In M. Jünger & V. Kaibel (eds.) *Integer Programming and Combinatorial Optimization 11*. Lecture Notes in Computer Science vol. 3509. Berlin: Springer.

[113] M. Oswald, G. Reinelt & D.O. Theis (2006) On the graphical relaxation of the symmetric traveling salesman polytope. To appear in *Math. Program.*

[114] M.W. Padberg & M. Grötschel (1985) Polyhedral computations. In E. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shmoys (eds.). *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, 307–360.

[115] M.W. Padberg & S. Hong (1980) On the symmetric travelling salesman problem: a computational study. *Math. Program. Study*, 12, 78–107.

[116] M.W. Padberg & M.R. Rao (1974) The travelling salesman problem and a class of polyhedra of diameter two. *Math. Program.*, 7, 32–45.

[117] M.W. Padberg & M.R. Rao (1982) Odd minimum cut-sets and b-matchings. *Math. Oper. Res.*, 7, 67–80.

[118] M.W. Padberg & G. Rinaldi (1987) Optimization of a 532 city symmetric traveling salesman problem by branch-and-cut. *Oper. Res. Lett.*, 6, 1–7.

[119] M.W. Padberg & G. Rinaldi (1990) Facet identification for the symmetric traveling salesman polytope. *Math. Program.*, 47, 219–257.

[120] M.W. Padberg & G. Rinaldi (1990) An efficient algorithm for the minimum capacity cut problem. *Math. Program.*, 47, 219–257.

[121] M.W. Padberg & G. Rinaldi (1991) A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems. *SIAM Rev.*, 33, 60–100.

[122] C.H. Papadimitriou (1978) The adjacency relation on the traveling salesman polytope is NP-complete. *Math. Program.*, 14, 312–324.

[123] C.H. Papadimitriou & M. Yannakakis (1984) The complexity of facets (and some facets of complexity). *J. Comp. System Sci.*, 28, 244–259.

[124] M. Queyranne & Y. Wang (1990) *Facet-tree composition for symmetric travelling salesman polytopes*. Technical Report 90-MCS-001, Faculty of Commerce and Business Administration, University of British Columbia.

[125] M. Queyranne & Y. Wang (1991) *Composing facets of symmetric travelling salesman polytopes*. Technical Report, Faculty of Commerce and Business Administration, University of British Columbia.

[126] M. Queyranne & Y. Wang (1993) Hamiltonian path and symmetric travelling salesman polytopes. *Math. Program.*, 58, 89–110.

[127] M. Queyranne & Y. Wang (1995) Symmetric inequalities and their composition for asymmetric travelling salesman polytopes. *Math. Oper. Res.*, 20, 838–863.

[128] M.R. Rao (1976) Adjacency of the traveling salesman tours and 0-1 vertices. *SIAM J. Appl. Math.* 30, 191–198.

[129] G. Reinelt & K.M. Wenger (2003) Small instance relaxations for the traveling salesman problem. In D. Ahr, R. Fahrion, M. Oswald & G. Reinelt (eds.) *Selected Papers of the Int. Conf. on Oper. Res. 2003*. Springer, pp. 371–378.

[130] A. Schrijver (2003) *Combinatorial Optimization: Polyhedra and Efficiency*. Springer.

[131] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[132] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, Massachusetts, 1997.

[133] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing, 6:445–454, 1994.

[134] A. Martin. Integer Programs with Block Structure. Habilitationsschrift, Technische Universität Berlin, 1999.

[135] R. E. Gomory. Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of The American Mathematical Society* 64, 275–278, 1958.

[136] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, 1998.

[137] H. Marchand and L. A. Wolsey. Aggregation and mixed integer rounding. *Operations Research*, 49:363–371, 2001.

[138] A. Schrijver. On cutting planes. *Annals of Discrete Mathematics* 9, 291–296, 1980.

[139] G. L. Nemhauser and L. A. Wolsey. A recursive procedure for generating all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.

[140] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975.

[141] E. Balas and E. Zemel. Facets of the knapsack polytope from minimal covers. *SIAM Journal of Applied Mathematics*, 34:119–148, 1978.

[142] Z. Gu, G. L. Nemhauser and M. W. P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10:427–437, 1998.

[143] M. W. Padberg, T. J. Van Roy and L. A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.

[144] T. J. Van Roy and L. A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45–57, 1987.

[145] M. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[146] A. H. Land and A. G. Doig. An Automatic Method for Solving Discrete Programming Problems. *Econometrica* 28, 497–520, 1960.

[147] E. Balas, S. Ceria, M. Dawande, F. Margot and G. Pataki. OCTANE: A New Heuristic For Pure 0-1 Programs. *Operations Research* 49, 207–225, 2001.

[148] E. Balas and C. H. Martin. Pivot-And-Complement: A Heuristic For 0-1 Programming. *Management Science* 26, 86–96, 1980.

[149] F. Glover and M. Laguna. General Purpose Heuristics For Integer Programming: Part I. *Journal of Heuristics* 2, 343–358, 1997.

[150] F. Glover and M. Laguna. General Purpose Heuristics For Integer Programming: Part II. *Journal of Heuristics* 3, 161–179, 1997.

[151] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publisher, Boston, Dordrecht, London, 1997.

[152] T. Ibaraki, T. Ohashi and H. Mine. A Heuristic Algorithm For Mixed-Integer Programming Problems. *Mathematical Programming Study* 2, 115–136, 1974

[153] A. Løkketangen. Heuristics for 0-1 Mixed-Integer Programming. In P.M. Pardalos and M.G.C. Resende (ed.s) *Handbook of Applied Optimization*, Oxford University Press, 474–477, 200

[154] P. L. Hammer, E. L. Johnson and U. N. Peled. Facets of regular 0-1 polytopes. *Mathematical Programming*, 8:179–206, 1975.

[155] M. W. Padberg. Covering, packing and knapsack problems. *Annals of Discrete Mathematics*, 4:265–287, 1979.

[156] Z. Gu, G. L. Nemhauser and M. W. P. Savelsbergh. Lifted flow cover inequalities for mixed 01 integer programs. *Mathematical Programming*, 85:439–467, 1999.

[157] A. Løkketangen and F. Glover. Solving Zero/One Mixed Integer Programming Problems Using Tabu Search. *European Journal of Operational Research* 106, 624–658, 1998.

[158] M. Nediak and J. Eckstein. Pivot, Cut, and Dive: A Heuristic for 0-1 Mixed Integer Programming. Research Report RRR 53-2001, RUTCOR, Rutgers University, October 2001.

[159] M. Fischetti and A. Lodi. Local Branching. *Mathematical Programming* 98, 23–47, 2003.

[160] E. Danna, E. Rothberg and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming* 102, 71–90, 2005.

Acknowledgements

I would like to thank Prof. Andrea Lodi, Prof. Paolo Toth and all the Operations Research people at DEIS for their unending support, competence and friendliness.

I owe much to Prof. Adam Letchford, who helped me greatly on the intricacies of theorem proving and has been a bottomless vault of bad instances and intuitions.

I would also like to thank Klaus Wenger for letting me use his very good cactus construction code.

All this wouldn't have been possible without the loving and caring support of my wonderful family.