Mathematical Programming 51 (1991) 141-202
North-Holland

Solution of large-scale symmetric travelling salesman problems

Martin Grötschel*

Institut für Mathematik, Universität Augsburg, W-8900 Augsburg, Germany

Olaf Holland**

Forschungsinstitut für Diskrete Mathematik, Institut für Operations Research, Universität Bonn, W-5300 Bonn, Germany

Received 18 July 1988
Revised manuscript received 31 July 1989

In this paper we report on a cutting plane procedure with which we solved symmetric travelling salesman problems of up to 1000 cities to optimality. Our implementation is based on a fast LP-solver (IBM's MPSX) and makes effective use of polyhedral results on the symmetric travelling salesman polytope. We describe the important ingredients of our code and give an extensive documentation of its computational performance.

AMS Subject Classifications: 05C04, 05C45, 90C10.

Key words: Travelling salesman problem, cutting plane algorithms, polyhedral combinatorics.

Introduction

Developing theory for the travelling salesman problem (TSP) and solving TSP's has always been one of the central subjects of mathematical programming. The TSP has not only fascinated mathematical programmers, operations researchers, and economists. Now also physicists, engineers, biologists, and chemists get excited about this problem. Reasons for this are certainly the facts that the TSP is easy to state, it has very nice applications, but it is hard to solve.

This paper contributes to the solvability aspects of the TSP. We describe an algorithm and its implementation with which large-scale travelling salesman problems can be solved to optimality. We see our work in a long line of attempts to use linear programming techniques and exploit information about the facet structure of the travelling salesman polytope. The history of this approach — outlined in

* Supported by DFG-Schwerpunkt "Anwendungsbezogene Optimierung und Steuerung", Universität Augsburg, Germany.
** Supported by SFB 303 (DFG), Forschungsinstitut für Diskrete Mathematik, Institut für Operations Research, Universität Bonn, Germany.

Grötschel and Padberg (1985), Padberg and Grötschel (1985) — started with the seminal paper Dantzig, Fulkerson and Johnson (1954) and reached a temporary peak in Crowder and Padberg (1980), who solved a 318 city problem, the largest problem solved to optimality until recently.

About 1980 we decided to make more systematic use of the classes of facets known for the travelling salesman polytope and to design a more powerful cutting plane algorithm to solve TSP's to optimality. There were many stimuli for this project. Around this time the ellipsoid method came into focus (with a revival of cutting plane ideas), the importance of polynomial time separation algorithms was discovered (see, e.g., Grötschel, Lovász and Schrijver, 1981, 1988), and Padberg and Rao (1982) invented a fast separation algorithm for the perfect 2-matching problem. Moreover, the computing power available increased considerably so that we hoped to be able to at least double (in terms of the number of nodes) the size of problems that can be solved to optimality. Our goal was to get close to the 1000 city barrier.

For this reason, one of us made up a geographical problem with 666 cities, he thought to be very hard, as a challenge for our work. It turned out to be just that in many respects. For instance, not only the integrality stipulations caused difficulties. Some linear programs that arose were hard to solve, even for highly praised commercial LP-codes like IBM's MPSX. But we finally managed to overcome all these obstacles.

We report here about the result of our work over these last years. Of course, this was not a continuous effort, and there are much longer periods of neglect of the problem than actual design and coding phases. We now feel that a stage is reached where our code has attained its limits. We have achieved our initial goal to solve travelling salesman problems up to 1000 cities. This, though, is still not a routine matter and requires a substantial amount of computing time on large computers; see Section 5. Some of the initial steps of our code design, restricted to certain relaxations of the TSP, can be found in Grötschel and Holland (1985, 1987). A much more detailed documentation of the code and its design is Holland (1987).

Let us mention here that Manfred W. Padberg and Giovanni Rinaldi have, in the recent years, developed a cutting plane code for the TSP that is based on the same approach and uses very similar ideas, many of which have been outlined in Padberg and Grötschel (1985). The design and the "tricks" of their code have not been documented completely yet, though some of the important ingredients appeared in Padberg and Rinaldi (1987, 1990a,b). In fact, the announcement in Padberg and Rinaldi (1987a) that a 2392 city problem was solved to optimality is really breath-taking.

Before describing our work we would like to add a few "philosophical" remarks concerning the questions: "What are all these efforts good for?" "Isn't it better to stick to heuristics?"

Everybody knows that travelling salesman problems may come up in tremendous sizes in practice. For instance, Bland and Shallcross (1987) report about problems

from crystallography with up to 15 000 cities; we know drilling problems (for printed circuit boards) of up to 60 000 cities. These problems seem far out of reach for our present (exact) algorithmic machinery. For the time being, these sizes can only be handled (approximately) with fast heuristics. We also do not advocate to solve, e.g., certain practical 1000 city drilling problems by running our code for a couple of hours in order to save one minute of drilling time. But there are some large-scale instances where knowing exact optima is important.

Optimization tools should not be applied blindly. One has to estimate whether or not it pays to use them, whether exact or approximate methods are the appropriate tools. We view our work mainly as a contribution to the state of the art of exact problem solving using LP-techniques and cutting plane procedures combined with heuristics and branch & bound.

Beautiful structural and algorithmic theory has been developed in the recent years. If one considers mathematical programming as a branch of applied mathematics, this should not remain just theory, it has to be put to work. The implementation process is more than straightforward and — at times — frustrating work. Often new interesting and challenging theoretical problems arise that have to be solved. But most of all it is the justification and validation of our scientific approach. The challenge of our time is large scale and we have to enlarge our algorithmic toolbox in various ways. We should not only confine ourselves to simple heuristics. Even for really large scale problems exact optimization is sometimes possible (and necessary). In addition, if only good approximate solutions are needed, the approach described here can be used heuristically in many ways to obtain excellent upper and lower bounds.

There is a further reason for our work. When starting this project, we had in mind to show that polyhedral combinatorics is not only nice theory but also a powerful algorithmic approach. We believe that the findings presented in this paper and the computational results of many similar projects completed in the recent years corroborate our point of view.

1. Notation

We will briefly mention a few symbols and definitions needed in the sequel.

We denote *graphs* by $G = (V, E)$, where V is the *node set* and E the *edge set.* All our graphs are *simple*, i.e., contain no loops and no multiple edges. An edge e with *endnodes* i and j is denoted by $e = ij$. The (up to isomorphism) unique graph on n nodes where every two nodes are adjacent is called *complete* and is denoted by K_n. The node set of a complete subgraph of a graph is called a *clique.* If G is a connected graph and W is a node set such that its removal disconnects G then W is called an *articulation set.* A Hamiltonian cycle (a cycle that contains every node of the graph exactly once) is also called a *tour.*

For a graph $G=(V, E)$ and $W \subseteq V$, we write

$$\delta(W) := \{ij \in E \mid i \in W, j \in V \setminus W\} \quad (= \delta(V \setminus W)),$$

$$E(W) := \{ij \in E \mid i, j \in W\}.$$

The edge set $\delta(W)$ is called the *cut* induced by W. If $W=\{v\}$ we write $\delta(v)$ instead of $\delta(\{v\})$.

If E is a finite set, then $\mathbb{R}^E$ denotes the set of functions from E to $\mathbb{R}$. This set is a real vector space and can be viewed as the set of vectors $x=(x_e)_{e \in E}$ where each component is indexed by an element of E. If $x \in \mathbb{R}^E$ and $F \subseteq E$ we write $x(F)$ to denote the sum $\sum_{e \in F} x_e$. The *incidence vector* $\chi^F \in \mathbb{R}^E$ of $F \subseteq E$ is the vector defined by $\chi_e^F = 1$ if $e \in F$, $\chi_e^F = 0$ if $e \notin F$.

A set $P \subseteq \mathbb{R}^E$ is a polytope if it is the convex hull of finitely many points. An inequality $a^T x \leqslant \alpha$ is *valid* with respect to P if $P \subseteq \{x \in \mathbb{R}^E \mid a^T x \leqslant \alpha\}$. A valid inequality $a^T x \leqslant \alpha$ defines a *facet* of P if $H := \{x \in P \mid a^T x = \alpha\}$ has dimension one less than P. An important fact from polyhedral theory is the following. If $P \subseteq \mathbb{R}^E$ is a polytope then there are an equation system $Ax = b$ and an inequality system $Dx \leqslant d$ such that $P = \{x \in \mathbb{R}^E \mid Ax = b, Dx \leqslant d\}$, A has full row rank and each inequality of $Dx \leqslant d$ defines a facet of P.

Finally, the (symmetric) travelling salesman problem is the following. Given a complete graph $K_n = (V, E)$ and distances c_{ij} for each edge $ij \in E$. Find a tour T with $c(T)$ as small as possible. Without loss of generality, we will assume throughout the paper that *all distances c_{ij} are integral.*

2. A short summary of some polyhedral results

To avoid some trivial technicalities let us assume from now on that the number n of cities (or nodes of the complete graph K_n) is at least 6.

Given a complete graph $K_n = (V, E)$, the (symmetric) *travelling salesman polytope* Q_T^n is the convex hull of all incidence vectors of tours of K_n. Thus

$$Q_T^n = \text{conv}\{\chi^T \in \mathbb{R}^E \mid T \subseteq E \text{ is a tour}\}.$$

The interest in this polytope derives from the fact that the symmetric travelling salesman problem can be solved by solving $\min\{c^T x \mid x \in Q_T^n\}$ which — in some sense — is a linear program. The polytope Q_T^n has been the subject of intensive investigations. A quite complete summary of the results on Q_T^n published to date can be found in Grötschel and Padberg (1985). (Let us mention, though, that very recently D. Naddef and G. Rinaldi and S. Boyd and B. Cunningham (personal communication) have discovered large new classes of facet-defining inequalities for Q_T^n.) We will briefly describe those equations and inequalities valid for Q_T^n that will be used in the sequel.

The affine hull of Q_T^n is defined by the linearly independent equations

$$x(\delta(v)) = 2 \quad \text{for all } v \in V. \tag{2.1}$$

Thus $\dim(Q_T^n) = |E| - |V|$. The *trivial inequalities*

$$0 \leqslant x_e \leqslant 1 \quad \text{for all } e \in E \tag{2.2}$$

also define facets of Q_T^n as well as the *subtour elimination constraints* (see Grötschel and Padberg, 1979) introduced by Dantzig, Fulkerson and Johnson (1954)

$$x(E(W)) \leqslant |W| - 1 \quad \text{for all } W \subseteq V,\ 3 \leqslant |W| \leqslant n-3. \tag{2.3}$$

Using (2.1) one can see that an inequality $x(E(W)) \leqslant |W| - 1$ is equivalent (defines the same facet) to $x(E(V \setminus W)) \leqslant |V \setminus W| - 1$. This in turn is equivalent to the *cut constraint* $x(\delta(W)) = x(\delta(V \setminus W)) \geqslant 2$. So the system of cut constraints

$$x(\delta(W)) \geqslant 2 \quad \text{for all } W \subseteq V,\ 3 \leqslant |W| \leqslant n-3, \tag{2.4}$$

defines the same facets of Q_T^n as the system (2.3).

Let $H, T_1, \ldots, T_s$ be a system of subsets of V. The inequality

$$x(E(H)) + \sum_{i=1}^{s} x(E(T_i)) \leqslant |H| + \sum_{i=1}^{s} (|T_i| - 1) - \lceil \tfrac{1}{2} s \rceil \tag{2.5}$$

is called a *2-matching constraint* (introduced in Edmonds (1965) to give a complete description of the 2-matching polytope) if $H, T_1, \ldots, T_s$ satisfy

$$|T_i \cap H| = 1, \quad i = 1, \ldots, s, \tag{2.6a}$$

$$|T_i \setminus H| = 1, \quad i = 1, \ldots, s. \tag{2.6b}$$

(2.5) is called *comb constraint* if $H, T_1, \ldots, T_s$ satisfy

$$|T_i \cap H| \geqslant 1, \tag{2.6a'}$$

$$|T_i \setminus H| \geqslant 1. \tag{2.6b'}$$

Grötschel and Padberg (1979b) proved that a 2-matching constraint or a comb constraint defines a facet of Q_T^n if, in addition, the node sets $H, T_1, \ldots, T_s$ satisfy

$$s \geqslant 3 \text{ and } s \text{ odd}, \tag{2.6c}$$

$$T_i \cap T_j = \emptyset, \quad 1 \leqslant i < j \leqslant s. \tag{2.6d}$$

The following class of valid inequalities for Q_T^n, which contains all nontrivial facet-defining inequalities listed above, was introduced by Grötschel and Pulleyblank (1986).

A *clique tree* is a connected graph C composed of cliques that satisfy the following properties (in the following we shall always consider clique trees as subgraphs of K_n):

(i) The cliques are partitioned into two sets, the set of *handles* and the set of *teeth.*

(ii) No two teeth intersect.

(iii) No two handles intersect.

(iv) Each tooth contains at least two and at most $n-2$ nodes and at least one node not belonging to any handle.

(v) The number of teeth that each handle intersects is odd and at least three.

(vi) If a tooth T and a handle H have a nonempty intersection, then $H \cap T$ is an articulation set of the clique tree.

Grötschel and Pulleyblank (1986) showed that, for every clique tree C with handles $H_1, \ldots, H_r$ and teeth $T_1, \ldots, T_s$, the following *clique tree inequality* defines a facet of Q_T^n,

$$\sum_{i=1}^{r} x(E(H_i)) + \sum_{j=1}^{s} x(E(T_j)) \leqslant \sum_{i=1}^{r} |H_i| + \sum_{j=1}^{s} (|T_j| - t_j) - \tfrac{1}{2}(s+1) =: s(C), \tag{2.7}$$

where, for every tooth T_j, the integer t_j denotes the number of handles that intersect T_j. Note that the facet-defining comb inequalities are exactly the clique tree inequalities associated with clique trees with only one handle.

Setting

$$Q_S^n := \{x \in \mathbb{R}^n \mid x \text{ satisfies (2.1), (2.2), (2.3)}\}, \tag{2.8a}$$

$$Q_{2M}^n := \{x \in \mathbb{R}^n \mid x \text{ satisfies (2.1), (2.2), and the 2-matching constraints (2.5), (2.6a,b)}\}, \tag{2.8b}$$

$$Q_C^n := \{x \in \mathbb{R}^n \mid x \text{ satisfies (2.1), (2.2), and the comb constraints (2.5), (2.6a',b')}\}, \tag{2.8c}$$

$$Q_{CT}^n := \{x \in \mathbb{R}^n \mid x \text{ satisfies (2.1), (2.2), (2.7)}\}, \tag{2.8d}$$

we see that $Q_T^n \subseteq Q_{CT}^n \subseteq Q_C^n \subseteq Q_{2M}^n$ and $Q_T^n \subseteq Q_{CT}^n \subseteq Q_S^n$.

Our approach to solving $\min c^T x$, $x \in Q_T^n$, is to use linear programming relaxations that can be defined by the polyhedra Q_S^n, Q_{2M}^n, Q_C^n and Q_{CT}^n. We will see later that the linear program

$$\min\{c^T x \mid x \in Q_S^n \cap Q_{2M}^n\}$$

which has a number of constraints that is exponential in n can be solved in polynomial time (in theory) by the ellipsoid method. In practice it can be solved by a simplex-based cutting plane procedure with reasonable efficiency. We do not know how to solve linear programs of the form $\min\{c^T x \mid x \in Q_C^n\}$, $\min\{c^T x \mid x \in Q_{CT}^n\}$, or $\min\{c^T x \mid x \in Q_T^n\}$ in theory or practice efficiently but we are able to generate some comb and some clique tree inequalities through separation heuristics (see Section 4). Thus or LP-based attack on the TSP ends with an optimum solution x^* of a linear program

$$\min\{c^T x \mid x \in Q\},$$

where Q is a polytope that contains Q_T^n and is contained in $Q_S^n \cap Q_{2M}^n$. If x^* is the incidence vector of a tour, we are done; otherwise we resort to branch & bound.

3. Outline of the code

We have explained the "philosophy" of our polyhedral approach to the TSP above. It is, however, a nontrivial and time consuming task to make this idea work.

We sketch now the important basic ingredients of our code. Details on cutting plane generation will be presented in Section 4. We assume that an instance of the symmetric travelling salesman problem is given by the number n of cities and by distances $c_{ij} \in \mathbb{Z}$, $1 \leq i < j \leq n$. The code has the four stages indicated in Figure 3.1.

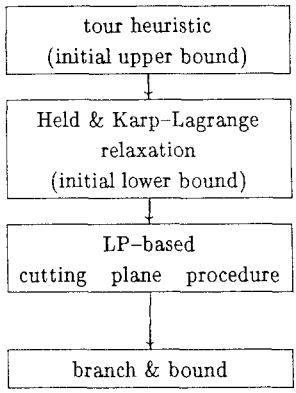


Fig. 3.1.

3.1. Preprocessing

The first two stages are mere preprocessing phases that help to speed up the cutting plane and the branch & bound phase. Their role is the following.

(1) *Tour heuristic.* By running heuristic procedures we generate several tours. The best tour found is later used to set up a starting basis for the initial LP of the cutting plane phase. The set $\hat{E}$ of edges that appear in at least one of the tours generated is stored. It is used to set up the initial linear program. Moreover, the length U of the best tour is utilized in the branch and bound phase as an upper bound.

We have chosen the following heuristic procedure. We first run the next neighbor heuristic 50 times starting with randomly chosen nodes. Each of the (different) tours produced this way is used as a starting tour for our implementation of the Lin-Kernighan heuristic — see Lin and Kernighan (1973). These methods are well-known and it is not necessary to describe the details here.

We would like to point out, however, that the Lin-Kernighan heuristic is rather time consuming. Running the procedure described in (1) in the standard way requires as much and sometimes even more time than the whole cutting plane and branch & bound method to be described later. Considerable parameter adjustments and coding efforts are necessary to make this heuristic run in acceptable time. On the

other hand, we (and of course others as well) have noticed that a good upper bound is as crucial as a good lower bound for a successful branch & bound phase. Thus, the quality of the upper bound achieved in this stage does have a serious impact on the overall performance of the algorithm.

Recall that our goal was to create a code that can solve symmetric travelling salesman problems of up to 1000 cities to optimality. This requires handling linear programs of up to half a million variables. We do not know any LP-code that can solve such linear programs. Thus, it is necessary to reduce the dimensions considerably. The reduction must, of course, be done in such a way that global optimality can still be proved. A first reduction step is based on the following procedure.

(2) *Held and Karp-Lagrange relaxation.* We solve the 1-tree relaxation described in Held and Karp (1970, 1971), using a standard subgradient algorithm. The largest value found in this procedure gives a lower bound L for the optimum tour length.

In theory, the solution of the Lagrange relaxation of the 1-tree problem gives the optimum value of $\min\{c^T x \mid x \in Q_S^n\}$. In practice, however, this value is rarely obtained through this procedure. By making good choices in the step length parameter etc. of the subgradient algorithm, good lower bounds for $\min\{c^T x \mid x \in Q_S^n\}$ can, in fact, be computed quickly.

(3) *First variable reduction.* Using the upper bound U found in (1), the lower bound L and the best 1-tree T found in (2) we can eliminate some edges based on a standard reduced cost criterion. For instance, if $e \in E \setminus (T \cap \delta(1))$, where 1 is the special node of the 1-tree, we know that $T \cup \{e\}$ contains exactly one cycle C not containing node 1. Suppose $\bar{c}_{ij} = c_{ij} + \lambda_i + \lambda_j$ is the cost of edge e, where λ_i, $i \in V$, is the set of best Lagrange multipliers. If f is an edge with largest cost $\bar{c}_f$ among the edges in $C \setminus \{e\}$ then edge e can be eliminated if $\bar{c}_e - \bar{c}_f > U - L - 1$. A similar criterion can be used for the edges $e \in \delta(1) \setminus T$.

Let $\bar{E} \subseteq E$ be the set of edges that can be eliminated due to these criteria, and let H be the tour giving the upper bound U. Then $E' := (E \setminus \bar{E}) \cup H$ is a set of edges that is guaranteed to contain at least one optimum tour of the original TSP. Thus we can restrict ourselves to considering the subgraph $G = (V, E')$ of K_n.

This finishes the description of the preprocessing phases. The variable elimination procedure described above is quite effective. Table 3.1 gives an overview of the results achieved in this way. The columns have the following meaning. "Problem (NUM)" is our name for the instance of the TSP. The number appearing in the name gives the number of cities of the TSP. (Details about the problems can be found in the appendix.) The second column shows the lower bound computed by the method described in (2). The third column reports the upper bound found in (1). The fourth column "%GAP" shows the maximal possible deviation (in percent) of the length U of the tour found in (1) from the optimum tour length. It is computed by means of the formula $(U - L) * 100/L$. The fifth column "Problem variables"

Table 3.1
Heuristic upper and lower bounds, reduction of variables

Problem (NUM)	Lower bound	Upper bound	%GAP	Problem variables	%VAR
17	2 047.7274	2 085	1.82	40	29.41
21	2 696.6135	2 707	0.39	31	14.76
24	1 265.2850	1 272	0.53	41	14.85
42	684.1273	699	2.17	188	21.84
48	4 953.3513	5 046	1.87	210	18.62
48H	11 425.9497	11 461	0.31	84	7.45
57	12 758.6891	12 985	1.77	332	20.80
70	669.0735	675	0.89	223	9.23
96	54 544.2939	55 209	1.22	557	12.21
100A	20 920 6091	21 282	1.73	681	13.76
100B	21 736.3089	22 141	1.86	847	17.11
100C	20 460.2047	20 749	1.41	619	12.51
100D	20 999.9530	21 294	1.40	617	12.46
100E	21 770.0528	22 068	1.37	602	12.16
100R	9 653.7772	9 690	0.38	181	3.66
120	6 902.3900	6 951	0.70	528	7.39
137	68 926.6767	69 853	1.34	1 196	12.84
200R	9 550.3396	9 653	1.07	721	3.62
202	39 502.0277	40 214	1.80	7 249	35.71
229	133 180.3029	134 666	1.12	3 879	14.86
300R	10 282.9372	10 424	1.37	1 742	3.88
318	31 182.3861	31 404	0.71	2 753	5.46
400R	9 496.0431	9 617	1.27	2 520	3.16
431	170 121.4302	171 778	0.97	24 354	26.28
442	5 038.7512	5 083	0.89	7 990	8.20
500R	9 161.2755	9 271	1.20	3 503	2.81
532	27 357.0196	27 829	1.73	43 282	30.64
600R	9 571.4626	9 732	1.68	6 654	3.70
666	292 188.0715	296 371	1.43	66 913	30.22
700R	10 120.1419	10 305	1.83	10 052	4.11
800R	10 094.6417	10 286	1.90	13 406	4.19
900R	9 995.6538	10 233	2.38	20 544	5.08
1000R	9 962.3615	10 156	1.94	20 774	4.16

contains the number $|E'|$ of variables remaining after the elimination procedure (3) has been executed. The sixth column "%VAR" shows the percentage of the number of remaining variables $|E'|$ compared to the number $|E|$ of original variables. E.g., in problem 500R only 2.81% of the variables are left for the final optimization step, while in problem 202 still 35.71% of the variables remain to be processed.

3.2. The cutting plane phase

This phase is the core of our algorithm. We enter it from the preprocessing phase with a subgraph $G' = (V, E')$ of $K_n = (V, E)$, a tour H whose length gives the upper bound U and with the set $\hat{E} \subseteq E'$ of edges that appeared in at least one of the

tours generated by the heuristic of (1). A flow chart of our cutting plane procedure can be found in Figure 3.2.

The aim of this phase is twofold. We want to produce a "very good" lower bound for the optimum tour value by LP-techniques and we want to set up a linear program whose 0/1-solutions contain the incidence vector of an optimum tour. To do this

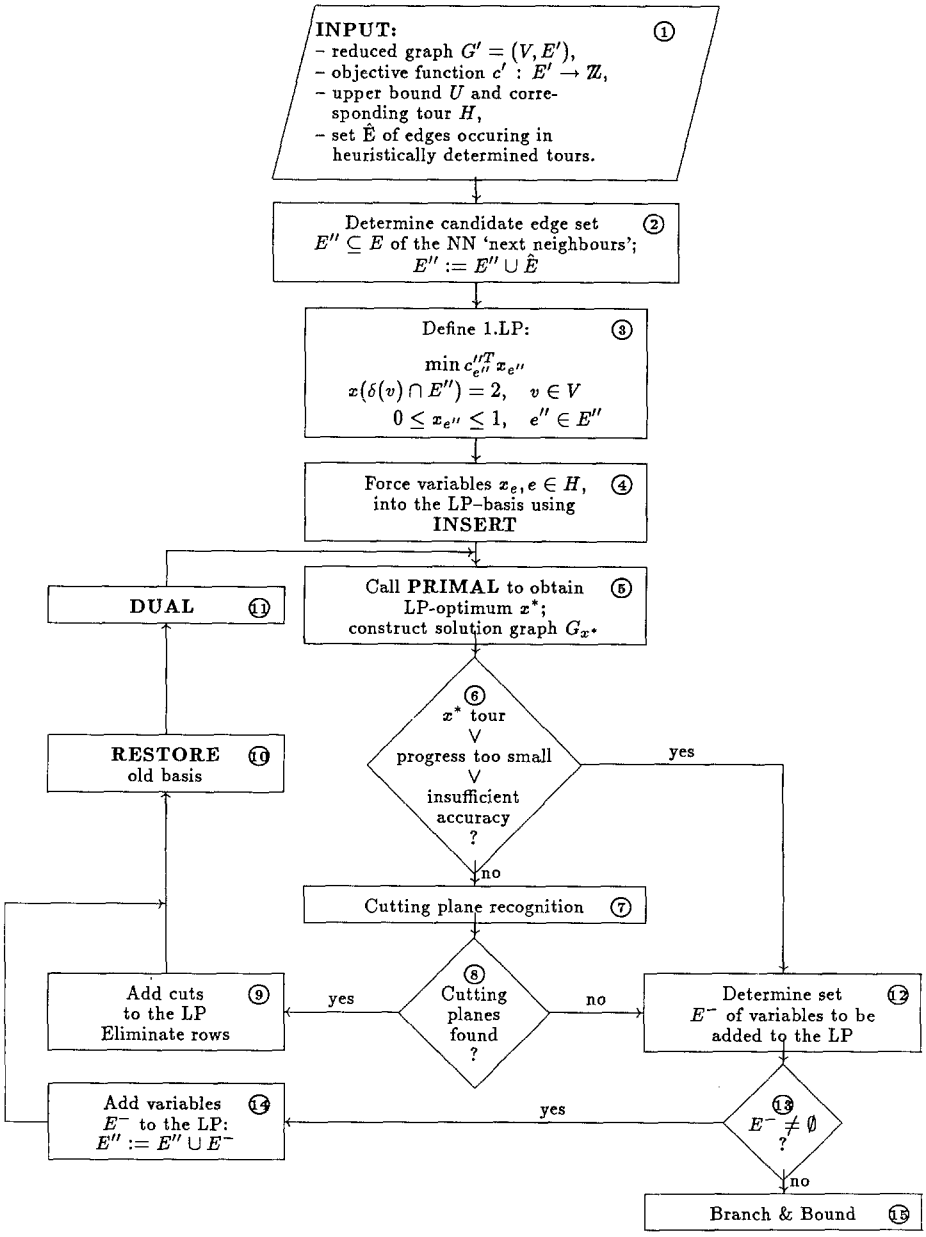


Fig. 3.2.

we first generate a set of edges $E'' \subseteq E$ and an inequality system $A''x'' \leq b''$, $x'' \in \mathbb{R}^{E''}$, such that the value of the linear program

$$\begin{aligned} &\text{minimize} && c''^{\mathrm{T}}x'' \\ &\text{subject to} && A''x'' \leq b'' \end{aligned} \tag{3.1}$$

is a true (and good) lower bound for the length of a shortest tour (c'' is the restriction of the vector c to the components E'').

The matrix A'' and the edge set E'' are not defined in advance. They are a result of the row and column generation scheme to be explained below. A'' has (by construction) the property that it can be extended in a canonical way to a matrix A with $|E|$ columns such that all inequalities of the system $Ax \leq b$ are either equations of the form (2.1) or define facets of Q_T^n. Moreover, $\min\{c''^{\mathrm{T}}x'' \mid A''x'' \leq b''\} = \min\{c^{\mathrm{T}}x \mid Ax \leq b\}$ holds.

Secondly A'' and E'' have the property that, by using reduced cost criteria, an extension of A'' to a matrix $\tilde{A}$ and an extension of E'' to a set of variables $\tilde{E}$ is possible such that the solution set of the 0/1-linear program

$$\begin{aligned} &\text{minimize} && \tilde{c}^{\mathrm{T}}\tilde{x} \\ &\text{subject to} && \tilde{A}\tilde{x} \leq \tilde{b}, \\ & && \tilde{x} \in \{0,1\}^{\tilde{E}}, \end{aligned} \tag{3.2}$$

contains the incidence vector of a shortest tour of the original problem. The 0/1-program (3.2) is the input to the branch & bound part of the algorithm, unless the solution (3.1) is the incidence vector of a tour (and we do not call branch & bound).

To achieve the goals described above, it would seem sensible to choose the edge set E' (of remaining variables) as the set of variables E'' to set up the linear program (3.1). Although our elimination procedure (3) is quite successful (see column %VAR of Table 3.1) the number of variables $|E'|$ is, in general, still much too large (e.g., $|E'| = 66\,913$ for the 666-city problem), even for fast commercial LP-solvers. So we decided to do the following.

(4) *Selection of initial variables.* We initially select a "candidate set" E'' of edges (of which we hope that they will contain an optimum tour) as follows. Depending on the parameter NN (we have used $0 \leq \mathrm{NN} \leq 10$) to be set before execution of the algorithm, we determine, for each node $v \in V$, a subset E_v of edges of $\delta(v)$ with cardinality NN having smallest length among all edges in $\delta(v)$ and set

$$E'' := \bigcup_{v \in V} E_v \cup \hat{E},$$

where $\hat{E}$ is the set of edges occurring in heuristically determined determined tours, see Section 3.1. This procedure is indicated in Box 2 of Figure 3.2. $\hat{E}$ is added to the "next neighbour edges" $\bigcup_{v \in V} E_v$ to guarantee the existence of a tour in

$G'' = (V, E'')$. The cutting plane procedure is initialized with the variable set E'', see Box 3 of Figure 3.2.

We now outline the LP solution techniques, the basics, and the main loop of the cutting plane part of our algorithm. These are indicated in Boxes $3, 4, \ldots, 11$ of Figure 3.2.

To solve the linear programs coming up we used IBM's package MPSX/370. This contains a quite fast LP-solver, though, for the application to be described in this paper, it does have some drawbacks that will be discussed later.

(5) *Initial LP and initial basis.* The initial linear program is defined in the standard fashion. We generate all degree constraints (2.1) and the upper and lower bounds (2.2). As outlined before we restrict the LP to the initial variables E'' defined in (4). Thus our first LP is of the form

$$\begin{aligned} &\text{minimize} && c''^{\mathrm{T}}x \\ &\text{subject to} && x(\delta(v) \cap E'') = 2 \text{ for all } v \in V, \\ & && 0 \leqslant x_{e''} \leqslant 1 \text{ for all } e'' \in E''. \end{aligned} \tag{3.3}$$

It is well known that, for every tour of the graph $G'' = (V, E'')$, one can determine a basis of (3.3) with the given tour as associated basic solution. We initialize our LP-solver by introducing the best tour H known at present as a starting basis using the MPSX-routine INSERT. (By the choice of E'', we have $H \subseteq E''$.) This process is indicated in Boxes 3 and 4 of Figure 3.2.

We now enter the main loop through Boxes $5, 6, \ldots, 11$ of Figure 3.2 and describe a general step.

To call the MPSX-routine PRIMAL in Box 5 we have to know a basis of our present LP. This is at hand in the first call due to (5). In a general step, a basis will be part of the output of the routine DUAL called in Box 11. PRIMAL determines an optimum solution x^* of the present LP. To process and analyse x^* we generate the graph $G_{x^*} := (V, E_{x^*})$ defined by

$$E_{x^*} := \{e \in E'' \mid x_e^* > 0\}. \tag{3.4}$$

The next step, Box 6 of Figure 3.2, consists of a couple of tests. We first check whether x^* is the incidence vector of a tour. If this is the case we go to the variable generation procedure of Box 12. Then we check whether the cutting plane procedure has "tailed off". We do this in the following way. Every tenth time we enter Box 6 we compare the present optimum LP-value γ^* with the optimum value γ of the linear program solved ten iterations (of the main loop) before. If $\gamma^* - \gamma \leqslant 1$ we feel that further cutting plane generations will not pay and exit from the loop to Box 12 of Figure 3.2. In a third test we check for numerical accuracy. MPSX offers some parameters to do this. If we feel that the present accuracy is insufficient, we leave

the loop for Box 12. If none of the tests made us go to Box 12 we continue with the cutting plane generation procedure of Box 7. The methods used here (and this is the key to success) will be described in Section 4.

The cutting plane generation phase of Box 7 determines inequalities of type (2.3), (2.5) or (2.7) that are violated by the present x^* and decides which of these inequalities should be added. If no new inequalities are provided by Box 7 we consider the present loop through the Boxes $5, 6, \ldots, 11$ finished and go from Box 8 to the variable generation Box 12. Otherwise we go to Box 9. In Box 9 we add all inequalities provided by the cutting plane procedure to the present LP using the MPSX-subroutine REVISE. Moreover, every fifth time we enter Box 9 we scan the rows of the present LP and eliminate all rows that are not satisfied with equality by the present optimum solution x^*, using REVISE again.

We now have a new LP for which we do not know a primal feasible basis. We do not want to solve this LP from scratch. Thus, we use the MPSX-subroutine RESTORE, Box 10, to initialize the basis with the optimum basis from the previous LP which is, after adding cuts, primal infeasible but still dual feasible, as a starting basis for the MPSX procedure DUAL; see Box 11. DUAL is not a dual simplex method. Its only purpose is to generate a "good" (very often optimal) primal feasible basis. When DUAL is finished we continue the main loop by going to Box 5.

The main loop is left through Boxes 6 or 8 to Box 12. In this case we have somehow generated inequalities (in a cutting plane fashion) in previous steps and currently finished looping through the Boxes $5, 6, \ldots, 11$. We end up with an optimum solution x^* of a linear program

$$\min\{c''^{\mathrm{T}}x'' \mid A''x'' \leqq b''\} \tag{3.5}$$

for which we cannot find any further cutting planes (see Box 8) or decided to stop the cut generation procedure (see Box 6). The optimum value of (3.5) is a lower bound for the optimum value of the TSP on the subgraph $G'' = (V, E'')$, but not necessarily for the optimum value of the original TSP. To check whether it is also a lower bound for $\min\{c^{\mathrm{T}}x \mid x \in Q_T^n\}$ we have to price out all the variables in $E \backslash E''$. Due to our elimination process (3) we need not touch any of the variables in $E \backslash E'$. Thus we can restrict ourselves to pricing out the variables in $E' \backslash E''$. Let $E^- \subseteq E' \backslash E''$ be the set of variables which might lead to an improvement. If $E^- \neq \emptyset$ we reset

$$E'' := E'' \cup E^-. \tag{3.6}$$

and return to the main loop. This procedure (pricing and variable addition) is indicated in Boxes 12, 13, 14 of Figure 3.2.

Initially we feared that when Box 12 is entered for the first time, E^- might be a very large set that would lead to LP's of unsolvable sizes. Empirically, in all problems we ran, E^- was of rather modest size. Thus we decided to add all variables in E^- to E''.

Let us mention at this point that — due to the large numbers of variables involved — pricing is quite a time consuming routine. One does not want to call it

too often. After some experiments we decided to trade space for time and to implement space-consuming data structures that allow us to execute pricing in reasonable time. (The details of this procedure are lengthy and boring and thus omitted.)

Of course, what we hope is that E^- is empty. In this case the optimum value of (3.5) is a true lower bound on the length of the shortest tour and our first goal is achieved. (The number of times we have to add variables clearly depends on the choice of NN. Even for NN = 0 it is quite moderate as can be seen in column SP of Table 5.1.)

If $E^- = \emptyset$ there are two possibilities: the current optimum solution x^* is fractional, or x^* is integral and thus a tour. From Box 6 of Figure 3.2 we know whether the optimum solution x^* of (3.5) is the incidence vector of a tour. If this is the case, we can stop with a globally optimum solution. This happened in 53 out of 128 runs on which we report later on. (With large real world problems, it never happened. Only small real world or random (small and large) problems were solved to optimality in this phase.) If x^* is not the incidence vector of a tour we must — in order to achieve our second goal — generate a 0/1-program of type (3.2). This procedure will be explained in the description of the branch & bound phase.

This finishes the outline of the LP cutting plane phase of our algorithm.

3.3. The branch & bound procedure

The flow chart shown in Figure 3.3 outlines our branch and bound procedure. We describe the procedure by explaining the boxes of this flow chart.

We enter this phase in Box 1 with a set of edges E'', the data of the linear program (3.5) $\min\{c''^{\mathrm{T}}x \mid A''x'' \leqslant b''\}$, the reduced costs $\bar{c}$, and an optimum (fractional) solution x^* of this LP. The optimum value $L := c''^{\mathrm{T}}x^*$ is a true lower bound for the optimum tour length. Moreover, we know from (1) a tour H of length U. We define

$$\mathrm{NB}_0 := \{e \in E'' \mid e \text{ nonbasic and } x_e^* = 0\},$$

$$\mathrm{NB}_1 := \{e \in E'' \mid e \text{ nonbasic and } x_e^* = 1\}.$$

In Box 2 we determine those edges in $E' \setminus E''$ that may appear in an optimum tour as follows. We generate the columns and the reduced costs $\bar{c}$ for all variables $e \in E' \setminus E''$. Then we set

$$\mathrm{VAR} := \{e \in E' \setminus E'' \mid \bar{c}_e \leqslant U - L - 1\} \tag{3.7}$$

and set in Box 3,

$$\tilde{E} := E'' \cup \mathrm{VAR},$$

$$\mathrm{NB}_0 := \mathrm{NB}_0 \cup \mathrm{VAR}.$$

The edges in $E \setminus \tilde{E}$ cannot be contained in an optimal tour. Thus they will never be

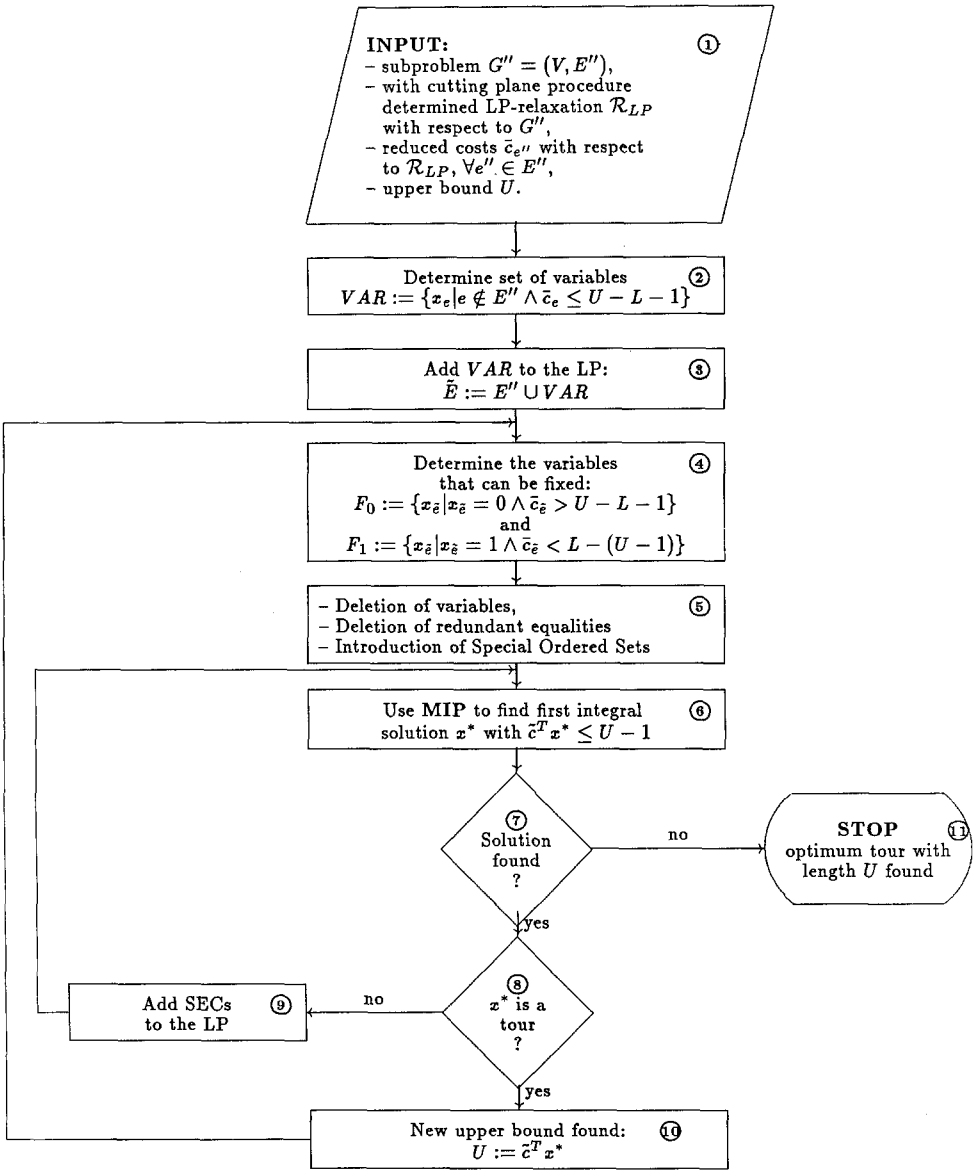


Fig. 3.3.

touched again. Then we generate the new linear program on the variable set $\tilde{E}$,

$$\text{minimize} \quad \tilde{c}^T \tilde{x}$$

$$\text{subject to} \quad \tilde{A}\tilde{x} \leq \tilde{b}$$

which has property that the 0/1-solutions of this LP contain an optimum tour. So the second goal mentioned in the beginning of Section 3.1 is achieved. From Box 3 we go to Box 4 of the flow chart of Figure 3.3.

Before describing Box 4 we want to mention that there is a problem here. The set VAR may be so large that there is no hope to finish the branch & bound phase in reasonable time. We have therefore added the following loop to the flow chart shown in Figure 3.3 not indicated in the chart. If $|\mathrm{VAR}| > 20\,000$ we do not add any variable at all in Box 3. We start the branch & bound phase with the old variables $\tilde{E} := E''$ and complete the whole process restricted to these variables. In this way we obtain a new upper bound U and go back to Box 2. The hope here is that the improved upper bound yields a much smaller set VAR.

The case explained above occurred in 5 out of the 128 runs described in this paper. In fact, in each of these five cases the first loop through the branch & bound phase produced a globally optimal tour. The new set VAR was therefore very small and thus the branch & bound phase had to be called only one more time.

We now come back to the flow chart and describe the main loop through Boxes 4, 5, ..., 10.

Box 4 contains a standard (global) variable-fixing procedure. For every edge $e \in \mathrm{NB}_0$ (the nonbasic variables at value 0), we check whether the reduced cost $\bar{c}_e$ is larger than $U-L-1$. If this is so we can fix the variable x_e forever at value 0. Similarly, for every $e \in \mathrm{NB}_1$, if $\bar{c}_e < L-(U-1)$ the variable x_e can be fixed at value 1. This way the sets F_0 and F_1 of fixed variables are defined.

The changes in the current LP that are induced by the variable fixing of Box 4 are executed in Box 5. The variables that have been fixed to zero or one are eliminated from the LP. The right hand sides are adjusted. If the right hand side of an equation of type (2.1) becomes zero, the whole equation is deleted. Some of the equations now have right hand side 1. We determine heuristically a subset of these equations that induce a system of special ordered sets (SOS). This information on a SOS-structure is handed over to MPSX since MPSX has special features that handle constraints of this type efficiently.

Now we enter Box 6 and call the mixed integer programming problem solver MIP, an additional feature of the MPSX system. MIP is a branch & bound code that offers only a few strategic choice parameters. So we have to use it as a black box and cannot influence the MIP decisions dynamically, add cutting planes on the run, etc. This is one of the clear drawbacks of the system. We set the MIP parameters in such a way that MIP outputs the first 0/1-solution x^* with objective function value less than or equal to $U-1$.

If MIP, in Box 7, reports that no such solution exists we can stop. The current best tour is optimal. If MIP produces a 0/1-solution we check in Box 8 whether x^* is the incidence vector of a tour. If this is the case (see Box 10) we have found a new upper bound $U := \tilde{c}^{\mathrm{T}} x^*$ that can be used to globally fix further variables. Thus we go back to Box 4 and continue the main loop.

If x^* is not the incidence vector of a tour it must be the incidence vector of a perfect 2-matching, due to the fact that the only 0/1-solutions of the linear system (2.1), (2.2) are incidence vectors of perfect 2-matchings. Hence we go from Box 8 to Box 9, where we determine (in a straightforward way) all cycles (=subtours) of

the corresponding perfect 2-matching. We generate the subtour elimination constraints induced by the node sets of these cycles, add these constraints to the current LP, and return to Box 6 where we call MIP again.

This is the overview of the branch & bound procedure and thus the end of the outline of our code for the TSP.

3.4. Some comments

Our outline shows mainly the strategic decisions. Needless to say that there are "infinitely many" little details that have to be considered, coded, tested etc. to make a complicated code like this work efficiently. There is no way to report about all this time-consuming work here. But we hope we have made our basic choices clear. We would have made some choices, in particular certain cutting plane generation, variable addition and deletion features, and the communication with the branch & bound code MIP in a different fashion if MPSX were an open code accessible by the user. In its present form MIP only allows a black box interface, and this is not adjustable to special needs.

A number of choices reported before (and in the following section) are quite arbitrary. For instance, why do we eliminate rows only every fifth time, why is tailing off checked every tenth time? We cannot give really convincing explanations for such decisions. We simply "played" with these (and other) parameters and set them to values that seem to produce decent results for which we could give an intuitive reason.

Let us explain one such decision. To keep the present LP small we should eliminate rows every time we run a new LP. However, our experiments showed that, after some iterations, the number of inequalities found in the cutting plane recognition phase is not very large and the number of rows that can be eliminated is not too large as well. Moreover, to find rows that can be eliminated we have to scan the whole inequality system and we have to actually eliminate them from the present system. To do the latter changes in the data structures, time consuming routines have to be called. So one has to balance the trade off between slightly longer running times of the LP-solver and time demanding data handling. Our experiments showed that elimination every fifth time works well in this respect.

Most of these parameters have been set by looking at them individually while keeping all the others fixed. Clearly, considering what statistics tells us, this is not the way to do it. One should design beforehand a series of experiments from which an optimal set of parameters can be derived. However, at present we do not see how one can satisfy the statistical assumptions necessary to derive the desired results.

4. The separation routines

We have mentioned before that our cutting plane procedure never works on the full set E of edges of the complete graph $K_n = (V, E)$ but on much smaller sets of

variables that are determined dynamically on the fly. For notational convenience we describe the separation routines for the case of the complete graph K_n. How to restrict what we do to subgraphs to K_n is obvious. Moreover, from a computational point of view, a restriction to sparse subgraphs of K_n results in a considerable speed-up of the running times of all routines. We describe now our procedures for finding violated subtour elimination constraints (2.3), violated 2-matching constraints (2.5), violated comb constraints (2.5), (2.6a',b'), and violated clique tree constraints (2.7). An overview of our strategy to call the various separation routines is shown in the flow chart of Figure 4.1. This flow chart is a blow up of Box 7 of Figure 3.2. Recall that we enter the cutting plane recognition phase from Box 6 of Figure 3.2.

The input to the cutting plane recognition phase is the optimum solution x^* of the current linear program. In Box 1 of Figure 4.1 we construct from x^* the "solution graph"

$$G_{x^*} = (V, E_{x^*}) \quad \text{with } E_{x^*} = \{ij \in E \mid x^*_{ij} > 0\}.$$

All routines described below use this graph. We will make use of the fact that x^* satisfies (2.1) and (2.2), i.e.,

$$x^*(\delta(v)) = 2, \quad v \in V, \qquad 0 \leqslant x^* \leqslant 1.$$

4.1. Finding violated subtour elimination constraints

Crowder and Padberg (1980) noticed how one can determine in polynomial time whether or not x^* violates one of the exponentially many subtour elimination constraints (2.3), or equivalently, one of the cut constraints (2.4). This procedure goes as follows.

For each edge $e \in E_{x^*}$, we consider the value x^*_e as a capacity. Then we compute a cut $\delta(W)$, $\emptyset \neq W \neq V$, of G^* with smallest capacity $x^*(\delta(W))$. If $x^*(\delta(W)) \geqslant 2$ then all cut constraints and all subtour elimination constraints are satisfied. Otherwise x^* violates the equivalent inequalities $x(E(W)) \leqslant |W| - 1$, $x(E(V \setminus W)) \leqslant |V \setminus W| - 1$, $x(\delta(W)) \geqslant 2$.

In such a case any one of these inequalities can be added to the current LP. We have decided never to add a cut constraint, and to add from the two possible subtour elimination constraints, the one induced by the smaller number of nodes. One reason is uniformity of the data structures we used (and do not want to explain in detail); the other is the empirically-observed fact that cut constraints in general tend to have more nonzero entries.

A minimum capacity cut of G_{x^*} can be found by applying the well-known Gomory-Hu procedure (see Gomory and Hu, 1961). The worst case running time of this method is $n-1$ times the running time of the max-flow algorithm used. So it is about $O(n^4)$. There are, however, ways to speed up the practical running time considerably. Much of the speed up comes from the fact that one does not apply the Gomory-Hu procedure to the original graph G_{x^*}. There are certain shrinking

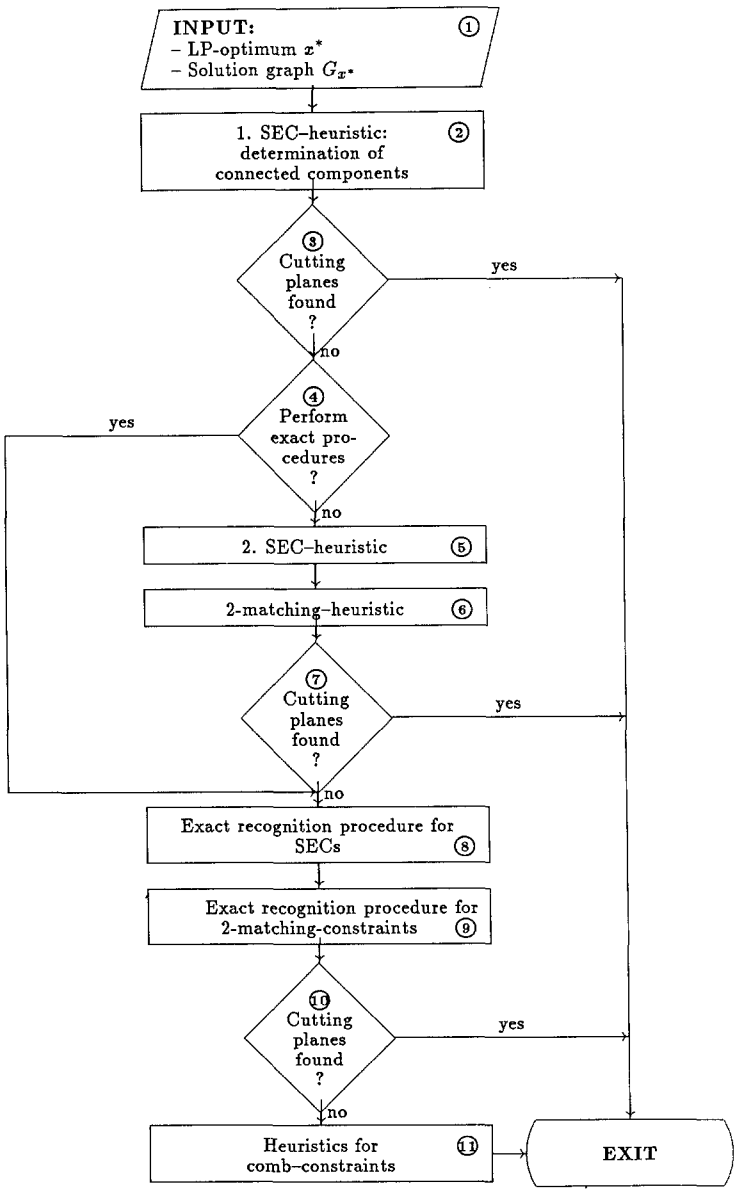


Fig. 4.1.

procedures, described for instance in Section 2.1 of Padberg and Grötschel (1985), that reduce G_{x^*} to a much smaller capacitated graph G'_{x^*} which has the property that G'_{x^*} contains a cut of capacity less than two if and only if G_{x^*} does. We have implemented this shrinking process which, in fact, may determine violated subtour elimination constraints without calling the Gomory-Hu method at all.

We have tested several max flow algorithms on problem instances coming up in this special case. It turned out that our implementation of the primal method described in Glover, Klingman, Mote and Whitman (1979) showed the best empirical running times. We use this program in our Gomory-Hu procedure.

The Gomory-Hu method requires a lot of data handling (new graphs have to be produced from the original graph by shrinking certain node sets etc.). We have tested several strategies to do this. In our present version we proceed, roughly, as follows. We do not shrink from scratch every time. After having determined a minimum weight cut in some current graph $\hat{G} = (\hat{V}, \hat{E})$ we immediately construct the (at most) two possible new graphs that can be derived from the new Gomory-Hu tree by shrinking certain node sets in $\hat{G}$ and store these two graphs for future processing on a stack and remove $\hat{G}$ from the stack. To save storage space we place the graph with the smallest number of nodes (but at least two nodes) on top of the stack. The success of this version of the Gomory-Hu procedure is evident from Table 5.2. It shows that the actual number of max-flow calculations is rather small.

We have recently learned that Padberg and Rinaldi (1990a) have invented an alternative strategy to compute a minimum capacity cut. Its empirical performance also is much better than the standard Gomory-Hu procedure.

Let us mention at this point that there are (at least) two options to run the exact separation routine for subtour elimination constraints described above. One can grow the full Gomory-Hu tree and read all violated constraints from the tree or one can stop the process as soon as a cut of capacity smaller than 2 is found. We tested both alternatives and found the latter to be inferior. Thus, whenever we enter our exact separation routine we compute the full Gomory-Hu tree. We generate all constraints that can be read from the tree. But only those that are violated by the current LP-solution by a value of some ε (we chose $\varepsilon = 0.001$) are considered to be "good" cutting planes that will be added to the linear program.

An ε-criterion for adding cutting planes like the one mentioned above will also be applied later for the other constraints. We observed empirically that MPSX sometimes ran into trouble when we added all violated constraints. (This has already been mentioned in the introduction). Since we cannot "open" MPSX we could not figure out whether the MPSX-problems were of numerical nature or due to degeneracy. The ε-criterion cured this disease.

We have also implemented two fast heuristics to determine violated subtour elimination constraints.

First subtour heuristic. Given the graph G_{x^*}, compute (by depth-first search) all connected components of G_{x^*}. Each component of G_{x^*} obviously yields a violated subtour elimination constraint.

This heuristic runs in $O(|E_{x^*}|)$ time. Compared to the running time of the whole algorithm it is unnoticable. Thus we run it always (see Box 2 of Figure 4.1) as soon as we enter the cutting plane recognition phase. If this heuristic finds at least one

subtour elimination constraint violated by x^* (see Box 3 of Figure 4.1) we stop the cutting plane recognition phase and return to the LP-routine (we enter Box 8 of Figure 3.2). This heuristic is quite successful in the first few iterations (say the first five calls of the separation program); afterwards it produces cutting planes only occasionally.

Our *second subtour heuristic*, Box 5 of Figure 4.1, is a variant of the preprocessing procedure for the Gomory-Hu algorithm that consists of a successive shrinking process. It is only applied in the first ten calls of the separation routine, since afterwards it turned out to be not too successful. Box 4 is used to switch the cutting plane recognition strategy after the tenth iteration.

Let us also mention that the (time consuming) exact separation routine for subtour elimination constraints is only called if the subtour and 2-matching heuristics fail to detect a violated inequality.

4.2. *Finding violated 2-matching constraints*

2-matching constraints are of the form

$$a^T x := x(E(H)) + \sum_{i=1}^{s} x(E(T_i)) \leqslant |H| + \sum_{i=1}^{s} (|T_i| - 1) - \lceil \tfrac{1}{2} s \rceil = s(C) \qquad (4.1)$$

where $H, T_1, \ldots, T_s \subseteq V$ are node sets satisfying (2.6a,b) and they define facets of Q_T^n if the node sets satisfy (2.6c,d) in addition. An exact polynomial-time separation algorithm for the class of 2-matching constraints has been designed by Padberg and Rao (1982). It is also based on the Gomory-Hu procedure.

In Grötschel and Holland (1987) we have outlined a fast (and successful) heuristic for finding violated 2-matching constraints and we have given a detailed description of our implementation of the Padberg-Rao procedure. Here we will only mention additional features that have been added to handle the TSP case.

The heuristic (see Box 6 of Figure 4.1) is called immediately after the second subtour heuristic has been called. If these two heuristics come up with at least one violated constraint, the cutting plane recognition phase terminates — see Box 7. The two heuristics are only run in the first ten calls of the separation routine. The count starts from fresh after each variable addition phase (Boxes 12, 13, 14 of Figure 3.2).

The exact separation algorithm for 2-matching constraints is called in Box 9 of Figure 4.1.We always grow a full Gomory-Hu tree to produce as many violated constraints as possible. We do, however, add a post-processing routine that does the following. For each violated 2-matching constraint, we compute $a^T x^* - s(C)$. If this quantity is smaller than 0.01, the cutting plane $a^T x \leqslant s(C)$ is ignored. Otherwise we check whether the 2-matching inequality defines a facet of Q_T^n (i.e., we test conditions (2.6c,d)). The Padberg-Rao procedure may, in fact, produce 2-matching inequalities that do not define facets of Q_T^n. The proof in Grötschel

(1977) that characterizes the facet-defining inequalities among the 2-matching constraints can be turned into a very simple algorithm that modifies a violated 2-matching constraint that does not define a facet of Q_T^n into a violated inequality that is either a facet-defining 2-matching constraint or a facet-defining subtour elimination constraint. We execute this modification and add the new cutting plane. If the exact separation routines for the subtour elimination and the 2-matching constraints find at least one violated inequality we terminate the cutting plane recognition phase through Box 10.

We have experimented with our code and stopped the two exact separation routines as soon as one violated inequality was discovered in order to save time in the cutting plane recognition phase. It turned out, however, that this led to much poorer overall performance. Many more LP's had to be set up; little progress in the objective function with tailing off phenomena occurred etc. For a similar reason we added the additional requirement that constraints to be added to the current LP have to be violated by at least a certain threshold ε (our choice was $\varepsilon = 0.01$ for 2-matching constraints and $\varepsilon = 0.001$ for subtour elimination constraints). If none of the separation routines described above produces a violated inequality we can conclude that x^* is "almost" contained in $Q_{2M}^n \cap Q_S^n$. (The "almost" creeps in because of the threshold values ε mentioned above.)

After treating subtour elimination and 2-matching constraints we try to make use of the further facet defining inequalities for Q_T^n listed in Section 2.

4.3. Finding violated comb constraints

It is not known whether the separation problem for the comb constraints (2.5), (2.6a',b'),

$$a^T x := x(E(H)) + \sum_{i=1}^{s} x(E(T_i)) \leqslant |H| + \sum_{i=1}^{s} (|T_i| - 1) - \lceil \tfrac{1}{2} s \rceil,$$

where $H, T_1, \ldots, T_s$ are subsets of V, can be solved in polynomial time. Hence we have invented some heuristic procedures with which some inequalities of this type can be discovered. The 2-matching constraints are the special cases where $|T_i| = 2$ for $i = 1, \ldots, s$.

Our basic idea is the following. Given the solution graph G_{x^*}, we manipulate G_{x^*} in such a way that some violated comb constraints are turned into violated 2-matching constraints. Then we apply the machinery for discovering violated 2-matching constraints described in Section 4.2. If violated (facet-defining) 2-matching constraints are detected this way we reverse the manipulation of G_{x^*} to obtain comb constraints violated by x^*.

We run the heuristics, to be described below, under the assumption that x^* satisfies all degree constraints (2.1), trivial constraints (2.2), subtour elimination constraints (2.3), and 2-matching constraints (2.5), (2.6a,b).

We now describe five basic transformations. For each transformation, we assume that a graph $N = (U, F)$ with capacities k_{ij}, $ij \in F$, is given.

Transformation 1 (F_1). If v has only two neighbours, say u and w, and if $k_{uv} = k_{vw} = 1$, we delete v from U and add the edge uw with the capacity $k_{uw} = 1$.

Transformation 2 (F_2, see Figure 4.2). Suppose $\{u, v, w\}$ is a clique of U with $k_{uv} + k_{vw} + k_{uw} = 2$ (i.e., the corresponding subtour elimination constraint is satisfied with equality). Suppose further that one of the edges, say uv, has capacity 1. Contract the two nodes u, v into one new node z (i.e., the edge uv is deleted, all edges with one endnode u or v will now have the endnode z. If parallel edges appear these are replaced by one edge whose capacity is the sum of the capacities of the parallel edges).

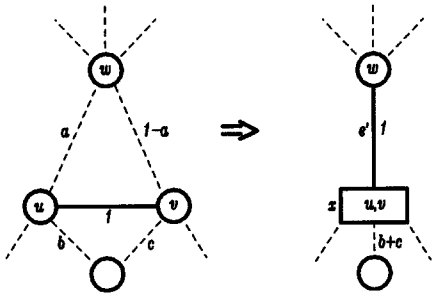


Fig. 4.2.

The result of this operation is that a 3-element clique that satisfies its subtour elimination constraint with equality is replaced by an edge of capacity 1.

Transformation 3 (F_3, see Figure 4.3). Suppose $\{u, v, w, x\}$ is a subset of U that satisfies

$$k(E(\{u, v, w, x\})) = 3.$$

Contract $\{u, v, w, x\}$ to a single node.

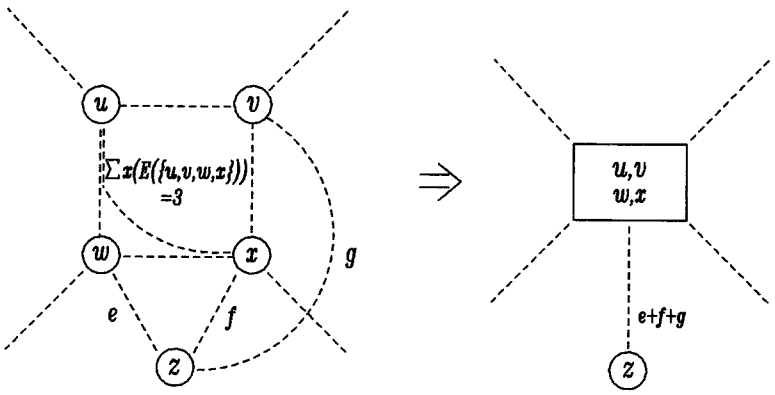


Fig. 4.3.

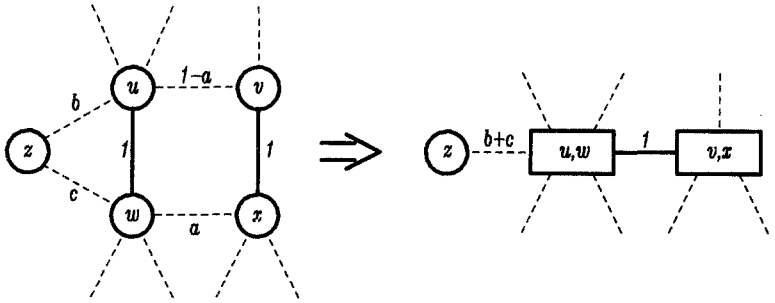


Fig. 4.4.

Transformation 4 (F_4, see Figure 4.4). Suppose the edges uv, vx, xw, wu form a cycle of N and have the following capacities

$$k_{uw} = k_{vx} = 1, \qquad k_{uv} = 1 - \alpha, \qquad k_{xw} = \alpha,$$

for some α with $0 < \alpha < 1$. Contract the node sets $\{u, w\}$ and $\{v, x\}$.

Transformation 5 (F_5). Suppose $\{u, v, w\}$ form a clique of U and $k_{uv} = 1$, $k_{uw} + k_{vw} \geq \frac{1}{2}$ is satisfied. Contract the node set $\{u, v\}$.

It should be clear that by using appropriate data structures all transformations described above can be undone. We will now describe how these transformations can be used to recognize violated comb constraints.

Generic comb heuristic (GCH).

Input: A graph $H = (U, F)$ with capacities k_e, $e \in F$, and a sequence $F_{i_1}, F_{i_2}, \ldots, F_{i_k}$ of the transformations $F_1, \ldots, F_5$.

Step 1. (a) Starting with H perform all transformations F_{i_1}, then all transformations $F_{i_2}, \ldots$, and finally all transformations F_{i_k} under the side constraint that an original node $u \in U$ participates at most once in a contraction process, a new node obtained by contraction and the endnodes of a new edge created in a transformation of type F_1 never participate in a contraction performed in the same iteration that generated them.

(b) If no transformation could be executed in (a) go to Step 3.

(c) Call the exact separation routine for 2-matching constraints. Stop the separation routine as soon as ten different violated 2-matching constraints are discovered. Undo the transformations to the very top level and create the corresponding violated comb constraints. Store these temporarily on a list. If the list contains more than 60 different comb inequalities go to Step 3.

Step 2. Call the new graph created in Step 1 H and return to Step 1.

Step 3. Determine the set of different comb inequalities on the list and remove those inequalities $a^T x \leq s(C)$ for which $a^T x^* - s(C) \leq 0.01$. *Simplify* (see below) the remaining comb inequalities and stop.

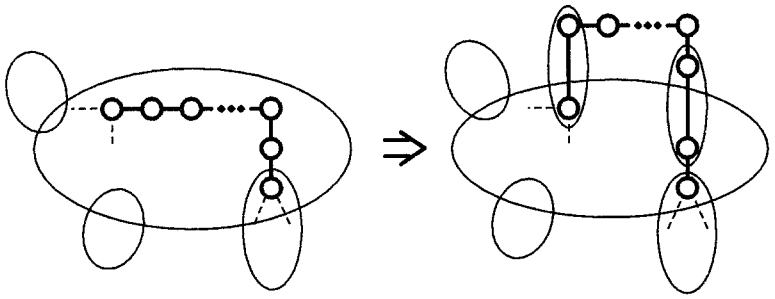


Fig. 4.5.

It may happen, for instance, that the comb inequalities created above contain long chains of edges of capacity 1 in the handle. In this case one can modify the comb as indicated in Figure 4.5.

This way a new violated comb inequality with a smaller handle and more teeth is created that has fewer nonzeros. There are further manipulations that can be performed to get inequalities with fewer nonzeros. We try a couple of these (but don't want to give the details) and call the whole process *simplification of comb inequalities.*

The generic comb heuristic GCH contains (again) many arbitrary-looking choices. Most of these choices grew out of experiments. Let us explain one. In Step 1(a) we add the condition that a newly created node may not participate in any further contraction in this step. The reason is that we observed that unlimited use of these transformations often shrinks a graph to nothing or that a violated comb constraint visible as a violated 2-matching constraint after one transformation disappears after a further transformation. Thus we decided to alternate between executing transformations and calling 2-matching separation and to iterate this process.

Now we describe how we set the free parameters $F_{i_1}, \ldots, F_{i_k}$, of our generic heuristic GCH.

Comb heuristic 1. Call GCH with the transformation sequence F_1, F_2, F_3.

Comb heuristic 2. Call GCH with the transformation sequence F_1, F_4, F_3.

Comb heuristic 3. Call GCH with the transformation sequence F_1, F_5, F_3.

Comb heuristic 4. Call GCH with the transformation sequence F_1, F_4, F_5.

We start each comb heuristic with capacities $k_e = x_e^*$. Figure 4.6 shows the solution graph G_{x^*} and a violated comb inequality detected by Comb Heuristic 1. Edges e with $x_e^* = 1$ are drawn with solid lines; edges with $x_e^* = \frac{1}{2}$ are drawn with dashed lines. Figure 4.7 shows a solution graph G_{x^*} and a violated comb inequality discovered by Comb heuristic 2.

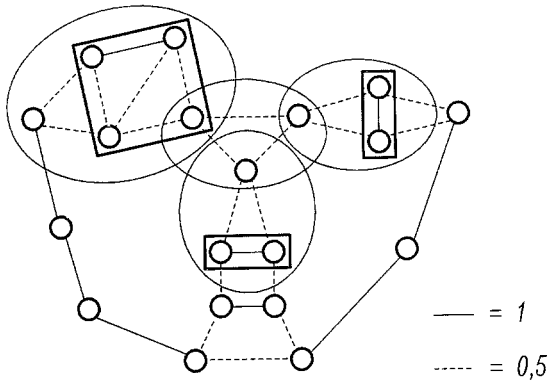


Fig. 4.6.

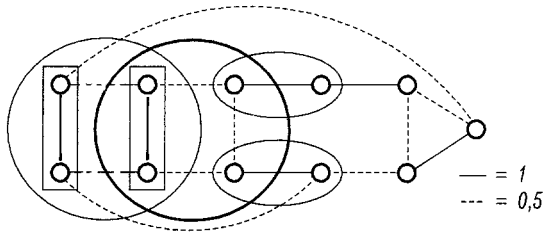


Fig. 4.7.

It is very hard to justify the choice of our transformations and the sequencing of these transformations convincingly. We have based our decisions in this respect on a careful analysis of fractional solutions arising in the cutting plane process and on extensive computational experiments.

The overall procedure for recognizing violated comb constraints works as follows. We call the Comb heuristics 1, 2, 3, 4 in this sequence. We only call the next heuristic if the previous ones found fewer than 60 violated comb constraints. All inequalities discovered in this process (see Box 11 of Figure 4.1) are then handed over to the LP-solver (see Box 8 of Figure 3.2).

4.4. Finding violated clique tree inequalities

It is unknown at present whether or not the separation problem for clique tree inequalities (2.7) can be solved in polynomial time. We have experimented with a number of heuristics, similar in spirit to the ones described in Section 4.3. We have concentrated our efforts on clique trees with exactly two handles and were able to discover some violated inequalities of this type. However, our procedures were not too successful; moreover, the clique tree inequalities increased the linear programming lower bound only insignificantly. Thus we decided to drop our clique tree recognition procedures from the current code for the time being. We believe that

more research in this area is necessary to come to a final conclusion about the usefulness of clique tree inequalities in a code as described here.

5. Computational results

We report here about a set of 31 symmetric travelling salesman problems and the execution of 129 runs on this set of data. The problems are named by a number and possibly a further letter. The number gives the number of cities. The additional letter is used to distinguish between problems and to denote randomly generated problems. If a name ends with a letter "R" it is a random problem; otherwise it is a real world problem. The problems named

17, 21, 24, 48, 96, 137, 202, 229, 431, 442, 666

are new problems. 442 is a drilling problem for a printed circuit board, all other problems are geographical problems, partly based on road distances, partly based on great circle distances on the globe. Data for these problems can be found in the appendix. The other problems are from the following sources:

42: Dantzig, Fulkerson and Johnson (1962);
48H: Held and Karp (1962);
57: Karg and Thompson (1964);
70: Smith and Thompson (1977);
100A-E: Felts, Krolak and Marble (1971);
120: Grötschel (1977);
318: Lin and Kernigham (1973);
532: Padberg and Rinaldi (1987).

(Remark: The 318-city problem is a hamiltonian path problem. We added an edge of length −9999 connecting the endpoints of the path to turn the problem in a TSP instance.)

The random problems have sizes 100, 200, ..., 1000. The random distances were generated uniformly from the range 1, ..., 5000. We have run all these problems with our code using the next neighbour parameter $NN \in \{0, 2, 5, 10\}$. There are two exceptions. The problems 532 and 666 were not run for $NN = 5$ and $NN = 10$. We expected prohibitive execution times.

The tests we report on have been performed on an IBM 3081D computer under the VM-operating system. As subprograms we used IBM's MPSX and MIP for the solution of the linear and integer programs. To communicate with these standard packages we used the ECL language feature of MPSX which is provided for PL/1. Thus most programs had to be implemented in PL/1.

In Tables 5.1 and 5.2 we report about the cutting plane phase (see Section 3.2) which uses the separation routines described in chapter 4. The first two columns of all tables that follow define the run by giving the name of the problem (column "Problem") and the number of next neighbours (column "NN") which has been

Table 5.1
Statistics on cutting plane phase

Problem	NN	Rows	Var	LP	Pivots	SP	LB	Time
17	0	28	39	7	337	4	2 085.0000	2*
17	2	28	40	7	406	4	2 085.0000	2*
17	5	28	57	4	148	1	2 085.0000	1*
17	10	28	105	4	167	1	2 085.0000	1*
21	0	21	28	4	167	4	2 707.0000	2*
21	2	21	32	2	61	2	2 707.0000	1*
21	5	21	68	1	27	1	2 707.0000	1*
21	10	21	128	1	25	1	2 707.0000	1*
24	0	26	41	3	129	2	1 272.0000	1*
24	2	27	48	4	251	2	1 272.0000	1*
24	5	28	80	3	172	1	1 272.0000	1*
24	10	27	150	2	97	1	1 272.0000	3*
42	0	58	139	15	1 729	4	699.0000	7*
42	2	56	113	7	818	4	698.0000	4
42	5	56	134	12	1 363	2	699.0000	5*
42	10	56	258	9	830	1	698.0000	4
48	0	102	127	38	6 545	5	5 041.0000	33
48	2	99	116	33	5 427	3	5 039.2500	20
48	5	91	154	26	3 867	2	5 039.5000	15
48	10	105	289	26	4 119	1	5 039.5000	19
48H	0	58	74	9	745	3	11 461.0000	3*
48H	2	63	84	10	1 060	3	11 461.0000	3*
48H	5	62	152	6	673	1	11 461.0000	2*
48H	10	62	300	6	692	1	11 461.0000	3*
57	0	79	122	26	4 839	5	12 955.0000	20*
57	2	107	112	20	2 804	2	12 955.0000	11*
57	5	121	186	26	4 397	2	12 955.0000	21*
57	10	97	369	20	3 363	1	12 955.0000	13*
70	0	117	171	41	10 450	4	675.0000	39*
70	2	157	164	37	10 028	4	675.0000	39*
70	5	155	224	22	5 222	2	675.0000	32*
70	10	140	414	31	7 616	1	675.0000	38*
96	0	269	287	80	46 462	5	55 113.7500	186
96	2	291	270	77	40 652	4	55 123.5000	179
96	5	238	316	60	26 581	2	55 052.3754	141
96	10	206	573	44	16 860	1	55 108.3000	101
100A	0	223	318	47	20 940	5	21 262.5000	61
100A	2	218	318	51	18 026	5	21 262.5000	55
100A	5	191	312	50	18 319	3	21 270.5000	73
100A	10	206	592	43	16 198	2	21 271.5000	67
100B	0	273	331	78	40 808	4	22 114.0000	183
100B	2	250	317	63	30 033	4	22 101.0000	110
100B	5	266	342	56	26 570	3	22 114.0000	136
100B	10	244	583	56	26 993	1	22 128.6667	124
100C	0	170	274	46	15 480	5	20 710.0000	32
100C	2	182	238	48	14 478	3	20 710.0000	41
100C	5	212	307	52	19 021	4	20 729.3333	70
100C	10	172	590	30	7 958	1	20 710.0000	39
100D	0	191	346	47	17 666	5	21 269.0000	46
100D	2	183	293	39	13 863	5	21 294.0000	42*

Table 5.1—continued

Problem	NN	Rows	Var	LP	Pivots	SP	LB	Time
100D	5	184	324	42	14 719	3	21 294.0000	46*
100D	10	157	577	21	5 896	1	21 269.0000	19
100E	0	290	333	86	42 372	4	22 032.2500	195
100E	2	273	308	98	46 716	4	22 039.1667	213
100E	5	294	327	64	37 475	2	22 033.0000	193
100E	10	268	580	63	28 094	1	22 032.2500	162
100R	0	101	178	3	602	2	9 663.0000	3*
100R	2	101	198	3	696	2	9 663.0000	4*
100R	5	101	310	2	311	1	9 663.0000	3*
100R	10	101	571	2	343	1	9 663.0000	3*
120	0	256	298	61	34 115	4	6 942.0000	109*
120	2	245	288	58	28 112	4	6 942.0000	104*
120	5	263	393	56	25 150	3	6 942.0000	184*
120	10	232	713	37	15 166	2	6 942.0000	99*
137	0	230	387	54	22 626	4	69 661.0000	116
137	2	234	365	55	23 571	4	69 733.8095	98
137	5	236	425	47	20 299	2	69 733.8095	104
137	10	257	806	44	18 296	1	69 753.4000	135
200R	0	227	411	19	7 611	3	9 583.4836	73
200R	2	236	433	17	6 402	1	9 583.5000	69
200R	5	220	631	12	4 787	1	9 583.4500	46
200R	10	255	1162	17	6 681	1	9 583.5000	126
202	0	457	469	91	74 860	5	40 154.3571	732
202	2	441	465	73	53 603	4	40 153.3000	472
202	5	469	663	71	58 181	2	40 157.5000	555
202	10	402	1263	55	36 651	1	40 151.6154	399
229	0	517	636	142	266 427	5	134 309.7728	1 098
229	2	553	588	119	188 456	4	134 355.6111	1 076
229	5	577	733	96	143 342	3	134 275.9782	994
229	10	579	1389	115	156 575	1	134 344.3203	925
300R	0	305	613	4	2 866	2	10 286.0000	12*
300R	2	305	644	4	3 435	2	10 286.0000	14*
300R	5	304	980	4	2 824	2	10 286.0000	14*
300R	10	304	1764	3	1 821	1	10 286.0000	16*
318	0	599	907	102	214 671	4	31 327.3857	1 167
318	2	617	903	84	183 704	3	31 333.0286	1 651
318	5	706	1145	66	93 186	4	31 319.7429	902
318	10	666	2020	69	92 848	2	31 327.7071	846
400R	0	403	792	4	3 211	2	9 501.0000	15*
400R	2	403	829	5	4 816	3	9 501.0000	18*
400R	5	403	1273	3	2 213	1	9 501.0000	16*
400R	10	403	2365	3	2 117	1	9 501.0000	20*
431	0	880	1117	166	544 179	7	171 172.2195	5 068
431	2	876	993	130	311 250	4	171 150.8823	3 601
431	5	846	1407	99	211 394	4	171 081.4944	2 648
442	0	803	963	124	219 843	5	5 065.2273	1 576
442	2	781	946	110	175 428	4	5 065.3333	1 409
442	5	806	1374	92	107 581	5	5 065.5357	783
442	10	756	2552	71	77 652	1	5 065.5000	1 025
500R	0	504	1025	5	6 267	3	9 166.0000	28*
500R	2	502	1057	3	3 616	2	9 166.0000	24*

Table 5.1—continued

Problem	NN	Rows	Var	LP	Pivots	SP	LB	Time
500R	5	504	1571	3	3 244	1	9 166.0000	27*
500R	10	502	2924	2	2 213	1	9 166.0000	28*
532	0	972	1410	172	598 756	6	27 633.5908	3 654
532	2	1011	1450	191	897 066	6	27 639.0466	6 004
600R	0	605	1224	6	9 309	2	9 579.0000	104
600R	2	607	1278	8	11 124	2	9 579.0000	132
600R	5	612	1907	8	20 052	2	9 579.0000	124
600R	10	610	3515	5	8 601	1	9 579.0000	123
666	0	1286	1644	154	610 026	6	293 972.7577	5 177
666	2	1363	1623	191	904 099	8	294 053.6900	10 298
666	5	1331	2202	141	543 450	5	294 035.9669	6 575
700R	0	705	1469	5	9 160	2	10 129.0000	173
700R	2	705	1526	5	8 740	2	10 129.0000	176
700R	5	703	2254	3	5 656	1	10 129.0000	45*
700R	10	705	4104	4	9 021	1	10 129.0000	164
800R	0	802	1729	4	11 813	3	10 101.0000	278
800R	2	808	1787	5	20 806	3	10 101.0000	301
800R	5	802	2597	3	17 544	2	10 101.0000	301
800R	10	802	4682	2	5 995	1	10 101.0000	261
900R	0	902	1951	3	12 005	2	10 006.0000	73*
900R	2	907	2011	6	20 052	2	10 006.0000	166*
900R	5	902	2892	2	6 475	1	10 006.0000	81*
900R	10	910	5272	5	24 506	1	10 006.0000	206*
1000R	0	1006	2129	9	24 558	2	9 971.0417	537
1000R	2	1006	2205	9	34 941	2	9 971.0417	546
1000R	5	1006	3222	9	37 792	2	9 971.0147	554
1000R	10	1006	5842	8	44 347	1	9 971.0417	554

used to define the first linear program (see Boxes 2 and 3 of Figure 3.2, also see Section 3.6). So, for most of the problems, four runs are documented.

We now describe Table 5.1. The column named "Rows" indicates the total number of rows of the last LP that had to be solved during the cutting plane phase. We delete rows according to the strategy described in Section 3.2. Thus this number does not necessarily give the maximum number of rows that occurred in an LP instance, but it is more significant as it indicates the complexity of the input for the branch & bound phase. The final number of inequalities (in the last LP to obtain the lower bound) is equal to "Rows" $-|V|$. For all problems, this number is less than $2|V|$ and for large problems it exceeds $|V|$ only in a few cases. Observe that, for the randomly generated problems, only a few (namely, 1-55) inequalities had to be added.

The number of variables of the last LP (that had to be solved during the run of the lower bound determination phase) is given in column "Var". For the problems with more than 200 nodes, 0.05% to 4% of the total number of variables of the original problem were needed to obtain the lower bound given in column "LB" which, on the average, is less than 0.1% off the optimum solution.

Table 5.2

Details on cutting plane recognition

Prob.	NN	CG	CD	DC	HS	H2M	HC	MHC	ES	MS	E2M	M2M
17	0	11	11	0	11	0	0	0	0	0	0	0
17	2	11	11	0	11	0	0	0	0	0	0	0
17	5	11	11	0	11	0	0	0	0	0	0	0
17	10	11	11	0	11	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0
21	2	0	0	0	0	0	0	0	0	0	0	0
21	5	0	0	0	0	0	0	0	0	0	0	0
21	10	0	0	0	0	0	0	0	0	0	0	0
24	0	2	2	0	0	2	0	0	0	0	0	0
24	2	3	3	0	1	2	0	0	0	0	0	0
24	5	5	4	0	3	2	0	0	0	0	0	0
24	10	3	3	0	3	0	0	0	0	0	0	0
42	0	17	16	0	7	6	4	370	0	0	0	47
42	2	15	14	0	10	0	0	116	2	4	3	49
42	5	16	14	0	7	4	2	292	2	4	1	53
42	10	16	14	0	7	4	0	71	2	10	3	40
48	0	102	54	28	10	16	48	1 570	8	181	20	724
48	2	82	51	8	11	21	25	1 090	9	114	16	478
48	5	68	43	8	17	12	17	533	5	110	17	515
48	10	86	57	6	17	12	24	416	8	129	25	601
48H	0	11	10	0	9	2	0	0	0	0	0	0
48H	2	18	15	0	16	2	0	0	0	0	0	0
48H	5	19	14	0	17	2	0	0	0	0	0	0
48H	10	19	14	0	17	2	0	0	0	0	0	0
57	0	69	22	29	15	14	34	766	6	33	0	149
57	2	59	50	1	17	6	8	327	6	56	22	244
57	5	82	64	0	15	10	23	859	8	124	26	490
57	10	54	40	0	15	10	8	255	7	50	14	295
70	0	106	47	26	18	22	37	2 281	9	198	20	964
70	2	144	87	19	22	16	58	1 697	10	244	38	1 020
70	5	119	85	21	22	12	51	1 413	10	125	24	600
70	10	108	70	12	22	10	31	1 295	10	223	35	1 016
96	0	315	173	80	26	23	142	7 542	29	1 500	95	4 884
96	2	362	195	84	30	22	167	6 397	31	1 453	112	4 672
96	5	261	142	50	24	8	103	4 764	33	1 303	93	4 038
96	10	216	110	40	24	8	66	2 381	35	738	83	2 498
100A	0	176	123	1	19	36	38	731	42	278	41	1 160
100A	2	164	118	1	31	45	17	492	35	304	36	1 317
100A	5	189	91	39	26	41	48	570	28	365	46	1 454
100A	10	157	106	1	29	22	25	395	26	421	55	1 770
100B	0	338	173	70	24	45	153	4 729	38	1 220	78	3 977
100B	2	261	150	42	32	24	87	2 434	39	1 035	79	3 394
100B	5	281	166	46	36	10	116	2 808	34	972	85	3 420
100B	10	249	144	39	35	10	85	1 649	28	973	91	3 588
100C	0	107	70	0	29	36	0	93	9	115	33	728
100C	2	135	82	6	25	38	19	652	19	178	34	986
100C	5	161	112	5	25	22	25	1 546	20	491	69	2 389
100C	10	111	72	8	23	18	10	204	11	190	49	1 064
100D	0	135	91	8	25	41	30	676	20	162	19	663
100D	2	107	83	1	30	27	12	709	14	78	24	440

Table 5.2—continued

Prob.	NN	CG	CD	DC	HS	H2M	HC	MHC	ES	MS	E2M	M2M
100D	5	141	84	3	28	29	11	433	40	204	33	749
100D	10	76	57	0	25	14	0	94	14	64	23	279
100E	0	426	190	66	23	29	147	8 686	39	856	188	4 354
100E	2	484	173	121	22	35	199	10 571	42	940	186	4 991
100E	5	399	194	50	24	12	125	4 683	36	1 159	202	4 698
100E	10	351	168	37	27	12	77	2 988	34	1 041	201	4 858
100R	0	1	1	0	1	0	0	0	0	0	0	0
100R	2	1	1	0	1	0	0	0	0	0	0	0
100R	5	1	1	0	1	0	0	0	0	0	0	0
100R	10	1	1	0	1	0	0	0	0	0	0	0
120	0	265	136	47	20	32	90	1 397	46	761	77	2 592
120	2	246	125	42	20	23	85	1 397	53	990	65	3 083
120	5	289	143	86	22	8	138	4 794	47	1 279	74	3 925
120	10	163	112	21	23	8	53	1 641	41	535	38	1 794
137	0	172	93	25	24	28	57	2 504	22	425	41	1 399
137	2	151	97	11	25	20	27	1 971	25	607	54	1 780
137	5	184	99	28	34	16	37	1 723	32	587	65	1 887
137	10	195	120	15	35	12	47	2 321	27	572	74	1 793
200R	0	47	27	19	6	6	27	687	1	334	7	799
200R	2	58	36	19	9	4	35	1 514	1	304	9	755
200R	5	41	20	18	9	4	19	432	1	159	8	386
200R	10	83	55	25	9	4	58	1 249	1	333	11	817
202	0	500	255	145	40	18	286	17 289	66	2 160	90	8 126
202	2	447	239	121	45	18	217	6 955	57	1 479	110	5 793
202	5	449	267	124	48	24	242	10 923	58	1 723	77	6 563
202	10	327	200	61	49	25	126	5 498	54	1 370	73	5 174
229	0	780	288	185	48	46	305	34 219	136	6 589	245	20 407
229	2	783	324	190	45	52	330	34 726	121	5 536	235	16 967
229	5	658	348	104	48	23	256	29 926	119	5 715	212	16 843
229	10	670	350	98	48	23	233	17 123	118	6 106	248	18 969
300R	0	7	5	0	7	0	0	0	0	0	0	0
300R	2	7	5	0	7	0	0	0	0	0	0	0
300R	5	7	4	0	7	0	0	0	0	0	0	0
300R	10	7	4	0	7	0	0	0	0	0	0	0
318	0	684	281	193	79	28	256	32 234	161	3 631	160	13 589
318	2	631	299	123	91	36	201	20 222	143	3 134	160	11 813
318	5	514	388	16	100	6	171	26 101	104	2 932	133	10 799
318	10	498	348	23	100	4	129	23 378	105	2 791	160	11 176
400R	0	3	3	0	1	2	0	0	0	0	0	0
400R	2	3	3	0	1	2	0	0	0	0	0	0
400R	5	3	3	0	1	2	0	0	0	0	0	0
400R	10	3	3	0	1	2	0	0	0	0	0	0
431	0	1062	449	268	68	53	460	82 058	228	12 284	253	37 886
431	2	1067	445	265	101	19	416	39 071	205	9 683	326	32 055
431	5	593	275	62	100	10	95	3 813	139	2 786	249	10 493
442	0	683	361	109	35	6	144	6 548	176	4 818	322	14 348
442	2	722	339	108	49	48	131	9 155	160	3 706	334	11 374
442	5	473	364	20	39	55	34	1 732	96	2 758	249	9 347
442	10	489	314	79	37	44	93	1 941	80	2 714	235	8 766
500R	0	6	4	0	4	2	0	0	0	0	0	0
500R	2	2	2	0	0	2	0	0	0	0	0	0

Table 5.2—continued

Prob.	NN	CG	CD	DC	HS	H2M	HC	MHC	ES	MS	E2M	M2M
500R	5	6	4	0	4	2	0	0	0	0	0	0
500R	10	2	2	0	0	2	0	0	0	0	0	0
532	0	1318	440	295	100	86	390	10 722	262	11 242	480	41 152
532	2	1474	479	380	95	67	509	21 059	376	16 618	427	58 837
600R	0	7	5	0	4	2	0	272	0	55	1	143
600R	2	10	7	0	4	4	1	878	0	121	1	310
600R	5	18	12	0	5	2	0	688	1	120	10	306
600R	10	10	10	0	1	2	1	538	2	81	4	205
666	0	1187	620	158	107	104	322	15 069	288	13 742	366	48 075
666	2	1521	697	386	128	74	598	93 317	299	20 682	422	71 003
666	5	1193	665	193	135	103	397	17 989	246	13 081	312	47 541
700R	0	6	5	0	2	4	0	254	0	12	0	46
700R	2	6	5	0	2	4	0	254	0	12	0	46
700R	5	3	3	0	1	2	0	0	0	0	0	0
700R	10	6	5	0	2	4	0	127	0	6	0	23
800R	0	6	2	0	6	0	0	708	0	81	0	213
800R	2	12	8	0	6	0	0	708	2	112	4	286
800R	5	6	2	0	6	0	0	472	0	54	0	142
800R	10	6	2	0	6	0	0	236	0	27	0	71
900R	0	3	2	0	3	0	0	0	0	0	0	0
900R	2	10	7	0	3	0	3	526	1	39	3	121
900R	5	3	2	0	3	0	0	0	0	0	0	0
900R	10	12	10	0	3	0	2	636	0	60	7	167
1000R	0	15	6	0	13	2	0	1 088	0	124	0	290
1000R	2	12	6	0	9	2	0	1 088	1	127	0	315
1000R	5	12	6	0	9	2	0	1 088	1	127	0	315
1000R	10	12	6	0	9	2	0	544	1	79	0	182

The number of linear programs which had to be solved (column "LP") exceeds only in a few runs $\frac{1}{2}|V|$. Column "Pivots" gives the total number of pivot-steps that had to be performed (summed over all LP calls). The 532- and the 666-city problems required the largest number of pivot-steps: about 900 000.

The number of times we had to enlarge the edge set of the subproblem considered so far, i.e., the number of times Box 14 in Figure 3.2 had to be executed, is given in column "SP". Let us mention that in most runs only a few variables (usually less than 10% of the number of variables given in column "Var") were added during the run to the initial LP defined in (5) of Section 3.2.

The time in seconds spent in the cutting plane phase of our algorithm, i.e., the total execution time of Figure 3.2, is given in column "Time". One of the runs for the 666-city problem took almost 3 hours while all the runs for the 1000-city problem terminated after less than 10 minutes.

A "*" in the final column indicates that the cutting plane phase produced a provably optimum tour, i.e., these problems did not enter the branch & bound phase.

In all four 700R runs the cutting plane phase terminated with the optimum tour value. But the optimum solution found was the incidence vector of a tour in only one of these cases.

Table 5.2 describes the behaviour of our cutting plane recognition procedure described in chapter 4. For each of the runs described above and identified by columns "Prob" and "NN", the following data have been recorded:

column "CG": total number of cutting planes generated;

column "CD": total number of cutting planes deleted (the last LP solved during the cutting plane phase contained CG − CD inequalities).

column "DC": deleted comb inequalities.

The columns "HS", "H2M" and "HC" give the number of heuristically determined subtour-elimination-, 2-matching- and comb constraints. Note that, according to our strategy, we run the second subtour elimination heuristic and the 2-matching heuristic only in the first ten calls (of each major iteration) of the separation routine. From our experiments we convinced ourselves that it pays off with respect to the overall efficiency to use "good" cutting planes. During the first ten calls of the separation routine we observed nicely-structured LP-solutions which allowed the heuristic determination of "good" cutting planes. Because subtour elimination constraints induced by connected components of the LP-solution graph are maximal with respect to the violation of the right hand side, the first subtour elimination heuristic has been called in all runs.

The number of cutting planes determined by our exact recognition procedures is given in column "ES" for the subtour elimination constraints and column "E2M" for the 2-matching constraints. As we grow in both cases a complete flow-equivalent tree to determine as many violated inequalities (of a certain quality, cf. chapter 4) as possible, the total number of performed max-flow calculations, reported in column "MS" for the subtour elimination procedure and in column "M2M" for the exact 2-matching procedure, is a significant indicator for the total computational effort that is necessary for the exact recognition procedures.

The number of max-flow calculations performed by the recognition procedure for violated comb inequalities is reported in column "MHC". The large numbers reported here justify our strategy to execute this heuristic only if all other recognition procedures fail.

For our present code there is no strong need to tune the cutting plane recognition procedure in the sense of Padberg and Rinaldi (1990a), because the time spent in these procedures (on the average about 25% of the execution time of the complete LP-phase) is approximately the same as the time required to update the linear programs (which is rather time-consuming with MPSX). But if the LP-package used is better suited for a row generation process than MPSX is, the total speed-up obtained by faster recognition procedures might be worth the higher programming effort.

Table 5.3 reports on the branch & bound procedure described in Section 3.3. Obviously, we did not run this procedure if the cutting plane phase ended with a

Table 5.3

Statistics on branch & bound phase

Prob.	NN	M	C	T	F_0	F_1	CR	D	S	SV	BV	Piv	Time	Opt
42	2	1	0	0	46	14	41	0	10	17	113	104	3	699
42	10	1	0	0	193	15	41	2	11	16	258	118	3	699
48	0	1	0	0	31	12	74	1	6	13	128	308	4	5 046
48	2	1	0	0	17	11	71	0	7	15	120	230	4	5 046
48	5	1	0	0	53	11	57	0	6	13	154	209	3	5 046
48	10	1	0	0	189	11	77	0	7	14	289	210	4	5 046
96	0	1	0	0	52	13	165	0	6	24	309	1 128	22	55 209
96	2	1	0	0	40	13	178	0	8	25	285	997	22	55 209
96	5	1	0	0	49	10	119	0	5	21	344	789	22	55 209
96	10	1	0	0	334	15	119	0	7	30	573	594	13	55 209
100A	0	1	0	0	102	24	162	1	14	41	319	687	11	21 282
100A	2	1	0	0	108	22	155	1	15	43	321	491	10	21 282
100A	5	1	0	0	103	22	130	0	15	40	317	525	10	21 282
100A	10	1	0	0	381	24	151	0	15	41	593	479	11	21 282
100B	0	1	0	0	105	20	198	0	10	33	340	820	18	22 141
100B	2	2	3	0	79	21	175	1	12	36	335	1 458	20	22 141
100B	5	1	0	0	109	21	193	0	10	33	353	870	20	22 141
100B	10	1	0	0	370	26	182	3	14	36	585	785	12	22 141
100C	0	2	1	0	60	19	91	0	11	35	288	1 002	9	20 749
100C	2	2	1	0	27	25	115	0	14	46	258	897	10	20 749
100C	5	2	1	0	93	23	145	1	11	32	310	957	11	20 749
100C	10	2	1	0	371	21	100	0	12	33	590	818	10	20 749
100D	0	1	0	0	140	25	133	0	14	41	352	571	10	21 294
100D	10	1	0	0	359	23	93	0	12	41	577	348	8	21 294
100E	0	1	0	0	104	25	225	2	11	38	349	873	17	22 068
100E	2	1	0	0	78	21	203	0	12	36	318	868	21	22 068
100E	5	1	0	0	86	25	225	1	11	35	336	1 429	24	22 068
100E	10	2	3	0	312	19	193	0	9	34	580	1 614	28	22 068
137	0	2	3	0	55	16	90	0	9	37	439	1 608	21	69 853
137	2	2	3	0	68	22	114	0	14	37	393	1 400	16	69 853
137	5	1	0	0	124	25	116	1	14	37	445	609	14	69 853
137	10	1	0	0	492	24	144	0	15	38	806	609	19	69 853
200R	0	2	0	1	275	81	194	13	36	78	573	4 128	32	9 589
200R	2	2	0	1	271	75	204	7	35	77	573	4 014	32	9 589
200R	5	2	0	1	378	83	177	16	34	74	672	4 075	28	9 589
200R	10	2	0	1	859	78	245	9	38	80	1 162	4 837	23	9 589
202	0	3	3	1	278	53	497	2	26	79	656	6 666	131	40 160
202	2	4	6	1	289	51	453	1	29	86	673	8 998	179	40 160
202	5	3	3	1	390	70	564	13	29	83	753	6 648	96	40 160
202	10	5	7	1	888	52	402	3	27	83	1 272	10 094	138	40 160
229	0	10	18	2	200	13	314	0	8	64	1 165	83 270	6 489	134 602
229	2	10	13	3	170	15	319	0	7	55	1 015	77 858	4 018	134 602
229	5	9	16	2	193	11	223	0	7	46	1 248	82 800	8 190	134 602
229	10	9	18	2	562	12	228	0	7	53	1 471	56 472	3 355	134 602
318	0	1	0	0	244	85	426	1	45	97	958	3 750	121	31 346
318	2	2	4	0	258	93	453	8	44	97	918	6 926	162	31 346
318	5	3	4	1	685	82	772	3	25	112	1 495	13 280	558	31 346
318	10	4	4	2	1 324	88	791	3	31	125	2 056	22 912	511	31 346
431	0	13	26	3	5 146	20	441	0	10	74	8 024	209 738	38 052	171 414
431	2	19	44	5	5 170	17	277	1	8	69	8 283	444 034	58 053	171 414
431	5	17	29	4	5 862	7	75	0	4	39	11 112	315 688	54 190	171 414
442	0	36	52	1	1 870	45	388	2	31	90	2 806	206 560	7 728	5 069
442	2	29	54	2	1 786	47	401	1	31	100	2 702	188 323	12 252	5 069
442	5	41	99	2	1 853	37	485	1	31	99	2 774	334 871	20 850	5 069
442	10	39	66	2	2 155	49	394	2	38	114	3 074	179 752	3 105	5 069
532	0	12	66	1	43	15	240	0	13	66	1 411	173 929	18 445	27 686

Table 5.3—continued

Prob.	NN	M	C	T	F_0	F_1	CR	D	S	SV	BV	Piv	Time	Opt
+532	0	9	21	0	75	36	389	0	20	125	2 657	197 854	45 081	27 686
532	2	15	45	1	64	22	296	0	18	90	1 451	154 201	15 766	27 686
+532	2	11	30	0	73	19	298	0	12	99	3 150	184 210	29 605	27 686
600R	0	5	4	2	5 505	251	477	53	145	233	6 320	9 174	161	9 580
600R	2	5	4	2	5 498	250	484	50	147	229	6 315	8 496	170	9 580
600R	5	4	1	2	5 538	269	511	69	141	223	6 342	11 171	157	9 580
600R	10	4	4	1	5 508	248	464	46	145	220	6 331	149 554	600	9 580
666	0	13	25	5	35	19	358	0	13	57	1 645	198 268	17 674	294 358
+666	0	3	2	0	55	22	215	0	14	93	5 189	46 663	9 855	294 358
666	2	16	27	7	43	23	491	0	15	61	1 624	326 517	33 395	294 358
+666	2	3	4	0	41	30	359	0	16	92	4 195	35 053	5 879	294 358
666	5	20	58	5	144	27	622	0	17	87	2 203	399 295	46 081	294 358
+666	5	3	4	0	166	31	317	0	20	128	4 578	32 138	4 531	294 358
700R	0	1	0	1	0	0	0	0	0	0	9 561	1 538	12	10 129
700R	2	1	0	1	0	0	0	0	0	0	9 570	1 466	12	10 129
700R	10	1	0	1	0	0	0	0	0	0	9 563	2 267	12	10 129
800R	0	2	0	1	11 851	359	630	90	200	291	12 930	6 012	232	10 102
800R	2	11	22	2	11 848	347	693	87	186	311	12 929	33 249	803	10 102
800R	5	10	18	2	11 883	355	681	99	172	297	12 958	43 932	816	10 102
800R	10	11	21	2	11 797	343	658	91	168	294	12 934	34 246	805	10 102
1000R	0	10	24	1	0	0	0	0	0	0	19 815	27 908	1 616	9 972
1000R	2	7	14	1	0	0	0	0	0	0	19 842	25 855	849	9 972
1000R	5	7	16	1	0	0	0	0	0	0	19 829	28 063	1 016	9 972
1000R	10	7	14	1	0	0	0	0	0	0	19 859	40 629	1 072	9 972

tour. This was the case in 51 runs out of 129 runs. In Table 5.3 every run where branch & bound was called is identified again by columns "Prob" and "NN".

The total number of calls to MIP (see Box 6 in Figure 3.3) is given in column "M". Thus MIP provided us with $M-1$ integral solutions, from which T (column "T") were tours. The number of subtour elimination constraints generated by Box 9 of Figure 3.3 is given in column "C".

During the run of the branch & bound procedure F_0 (F_1) variables have been fixed to 0 (1) by the procedure in Box 4 of Figure 3.3 and consequently could be deleted from the integer program, which initially (after the execution of Box 3 of Figure 3.3) contained "BV" variables. Fixing variables caused "CR" rows to be changed. From the initial set of the $|V|$ degree-equalities, "D" rows obtained a right hand side of zero and could therefore be deleted from the integer program.

Furthermore, fixing of variables allowed the introduction of S special ordered sets (see Box 5 of Figure 3.3), involving "SV" variables. As mentioned above, the determination of these sets has been rather arbitrary and was not optimized with respect to the number of variables involved or any other objective function.

The total number of pivot operations performed is reported in column "Piv". To obtain and "prove" the optimality of the shortest tour found which has length "Opt", "Time" seconds were needed.

As described in Section 3.3, if the number of variables that would have to be added according to Box 3 of Figure 3.3 exceeded 20 000, we did not add any variable at all. We first executed the branch & bound phase on the smaller set of variables to obtain a better upper bound. This has been necessary for the following runs: Prob = 532, NN $\in \{0, 2\}$; and Prob = 666 and NN $\in \{0, 2, 5\}$. The second run with the improved upper bound, which turned out to be the overall optimum solution in all of our runs, is marked by the + sign in Table 5.3.

6. Conclusions

We would like to mention a few things that might improve the approach presented here.

We are sure that the separation routines can still be improved with respect to speed and success in finding cutting planes. In particular, there is much to be done concerning comb constraints, clique tree inequalities and further classes of inequalities not mentioned here.

A basic design error we made was to use a black box LP-solver. Commercial LP-solvers like MPSX are certainly much faster than any such code we can come up with. That is why we used it. But from a certain size on, in using MPSX most time is wasted by communicating between the various parts of the code, setting up and revising data structures. LP-solving, cutting plane addition, and branch & bound have to be married in order to be really successful.

For instance, the branch & bound routine MIP of MPSX gives only very global control to the user, and cutting planes cannot be added on the run. We tried, thus, to write our own branch and bound environment. But although our branching trees were considerably smaller we could not beat MIP in overall performance because we had to access MPSX and its subroutines through slow communication interfaces.

Padberg and Rinaldi (1987) use R. Marsten's XMP code. This is certainly a slower LP-solver than MPSX, but it can be adjusted to special needs. We believe that this contributed significantly to the greater success of this code. It may, in fact, well be that LP-solvers have to be run with special column and row selection rules for LP-relaxations of combinatorial optimization problems in order to be really efficient. Investigations of this type still have to be done.

It also remains to be checked whether the new interior point methods can help to solve combinatorial optimization problems. What we need are efficient ways to add and delete rows and columns dynamically and to resolve LP's adjusted this way frequently. The rumours that interior point methods are particularly successful for solving large scale linear programs might give rise to hopes that also the solvability of problems like the TSP can be pushed an order of magnitude further. There is still much to be done.

7. Appendix. Data and optimal solutions of new problems

The problems 17, 21, 24 and 48 are derived from road maps. The corresponding distance matrices are defined in the sequel as upper triangular matrices with main diagonal (=0) stored columnwise.

The problems 96, 137, 202, 229, 431 are subproblems of the 666-city-problem, which is defined by the list of coordinates in Table 7.5 and the PASCAL procedure of Table 7.6 for the determination of great circle distances. The subproblems are defined by the following coordinates of the 666-city-problem:

96: 139–234, 137: 2–138, 202: 235–436, 229: 437–665, 431: 235–665.

The 442-node problem is a drilling problem for a printed circuit board and is defined by the integral distances $d(x, y)$ obtained from the coordinates (rowwise notation) in Table 7.7 by the following formula:

$$d(x, y) := \mathrm{SQRT}((x_1 - x_2)^2 + (y_1 - y_2)^2) + 0.5,$$

where SQRT denotes the FORTRAN IV real * 4 square root function.

Table 7.1

Data of the 17-city-problem

0	633	0	257	390	0	91	661	228	0	412	227	169	383	0	150	488	112
120	267	0	80	572	196	77	351	63	0	134	530	154	105	309	34	29	0
259	555	372	175	338	264	232	249	0	505	289	262	476	196	360	444	402	495
0	353	282	110	324	61	208	292	250	352	154	0	324	638	437	240	421	329
297	314	95	578	435	0	70	567	191	27	346	83	47	68	189	439	287	254
0	211	466	74	182	243	105	150	108	326	336	184	391	145	0	268	420	53
239	199	123	207	165	383	240	140	448	202	57	0	246	745	472	237	528	364
332	349	202	685	542	157	289	426	483	0	121	518	142	84	297	35	29	36
236	390	238	301	55	96	153	336	0									

Table 7.2

Data of the 21-city-problem

0	510	0	635	355	0	91	415	605	0	385	585	390	350	0	155	475	495
120	240	0	110	480	570	78	320	96	0	130	500	540	97	285	36	29	0
490	605	295	460	120	350	425	390	0	370	320	700	280	590	365	350	370	625
0	155	380	640	63	430	200	160	175	535	240	0	68	440	575	27	320	91
48	67	430	300	90	0	610	360	705	520	835	605	590	610	865	250	480	545
0	655	235	585	555	750	615	625	645	775	285	515	585	190	0	480	81	435
380	575	440	455	465	600	245	345	415	295	170	0	265	480	420	235	125	125
200	165	230	475	310	205	715	650	475	0	255	440	755	235	650	370	320	350
680	150	175	265	400	435	385	485	0	450	270	625	345	660	430	420	440	690
77	310	380	180	215	190	545	225	0	170	445	750	160	495	265	220	240	600
235	125	170	485	525	405	375	87	315	0	240	290	590	140	480	255	205	220
515	150	100	170	390	425	255	395	205	220	155	0	380	140	495	280	480	340
350	370	505	185	240	310	345	280										
105	380	280	165	305	150	0											

Table 7.3

Data of the 24-city-problem

0	257	0	187	196	0	91	228	158	0	150	112	96	120	0	80	196	88
77	63	0	130	167	59	101	56	25	0	134	154	63	105	34	29	22	0
243	209	286	159	190	216	229	225	0	185	86	124	156	40	124	95	82	207
0	214	223	49	185	123	115	86	90	313	151	0	70	191	121	27	83	47
64	68	173	119	148	0	272	180	315	188	193	245	258	228	29	159	342	209
0	219	83	172	149	79	139	134	112	126	62	199	153	97	0	293	50	232
264	148	232	203	190	248	122	259	227	219	134	0	54	219	92	82	119	31
43	58	238	147	84	53	267	170	255	0	211	74	81	182	105	150	121	108
310	37	160	145	196	99	125	173	0	290	139	98	261	144	176	164	136	389
116	147	224	275	178	154	190	79	0	268	53	138	239	123	207	178	165	367
86	187	202	227	130	68	230	57	86	0	261	43	200	232	98	200	171	131
166	90	227	195	137	69	82	223	90	176	90	0	175	128	76	146	32	76
47	30	222	56	103	109	225	104	164	99	57	112	114	134	0	250	99	89
221	105	189	160	147	349	76	138	184	235	138	114	212	39	40	46	136	96
0	192	228	235	108	119	165	178	154	71	136	262	110	74	96	264	187	182
261	239	165	151	221	0	121	142	99	84	35	29	42	36	220	70	126	55
249	104	178	60	96	175	153	146	47	135	169	0						

Table 7.4

Data of the 48-city-problem

0	593	0	409	258	0	566	331	171	0	633	586	723	874	0	257	602	522
679	390	0	91	509	325	482	598	228	0	412	627	506	663	227	169	383	0
378	755	634	791	397	175	349	167	0	593	416	564	721	271	445	509	293	429
0	150	598	414	571	488	112	120	267	233	541	0	659	488	630	787	205	511
575	304	470	76	607	0	80	566	382	539	572	196	77	351	317	563	63	629
0	434	893	699	856	524	231	405	303	138	595	289	606	373	0	455	417	433
590	313	304	371	228	394	158	399	224	425	530	0	134	583	399	566	530	154
105	309	275	575	34	638	29	298	434	0	649	945	824	981	446	423	620	357
280	649	504	648	588	416	584	564	0	259	364	180	337	555	272	175	338	446
403	264	469	232	549	265	249	656	0	505	354	110	70	819	619	421	602	730
660	509	728	478	795	529	494	920	276	0	710	117	375	354	679	693	626	720
848	533	715	610	683	986	534	700	1038	481	345	0	488	784	663	820	289	262
459	196	119	488	343	502	427	255	423	385	161	495	759	877	0	353	641	520
677	282	110	324	61	125	353	208	364	292	261	288	250	315	352	616	734	154
0	324	275	91	248	638	437	240	421	549	486	329	552	297	614	348	314	739
95	187	392	578	435	0	605	287	431	588	313	445	520	470	598	143	610	215
577	734	144	595	788	352	527	404	627	484	385	0	372	229	39	196	686	485
288	469	597	511	397	578	345	662	396	361	787	143	135	346	626	483	54	377
0	330	484	361	518	378	119	260	150	278	323	174	389	276	414	185	207	468
193	475	577	307	164	276	326	324	0	581	877	756	913	370	355	552	289	212
581	436	571	520	348	516	478	84	588	852	970	93	247	671	720	719	400	0
154	460	276	433	612	298	63	453	419	460	190	526	158	475	322	175	690	126
372	577	529	396	191	471	239	250	622	0	70	523	339	496	569	191	27	346
312	515	83	589	47	368	385	68	583	189	435	640	422	287	254	534	302	249
515	115	0	606	183	216	147	715	719	522	703	831	549	611	615	579	896	546
596	1021	377	139	209	860	717	288	416	242	558	953	473	536	0	585	427	563
720	179	437	501	196	362	80	532	108	558	498	163	567	552	395	659	544	391
256	478	154	526	318	484	452	515	556	0	544	840	719	876	311	318	515	252
175	508	399	494	483	311	479	441	154	551	815	933	65	210	634	683	682	363

Table 7.4—continued

77	585	479	916	399	0	496	525	595	751	147	253	468	85	251	208	351	236
435	387	162	393	441	427	691	646	280	145	509	249	558	239	373	538	430	654
128	336	0	317	289	105	262	631	430	233	414	542	479	332	545	290	607	341
307	732	88	201	406	571	428	21	407	68	269	664	184	247	302	471	627	503
0	648	68	316	362	584	598	564	625	753	418	653	484	621	891	415	638	943
395	412	95	782	639	333	285	287	482	875	515	578	209	425	838	523	347	0
211	660	476	633	466	74	182	243	171	489	66	555	150	227	351	108	432	326
572	777	271	184	391	492	439	166	364	252	145	673	438	327	327	384	715	0
475	137	295	452	437	428	391	452	580	271	480	337	448	718	268	465	770	222
391	254	609	466	255	138	241	309	702	342	405	287	278	665	376	277	167	542
0	654	151	319	266	755	767	570	751	879	561	659	627	627	944	558	644	1069
425	262	103	908	765	336	428	290	606	1001	521	584	122	568	964	666	350	169
721	299	0	710	239	487	546	616	660	626	687	815	443	715	509	683	953	440
700	1005	457	583	279	844	701	490	310	458	544	937	577	640	393	450	900	548
512	179	777	229	353	0	585	135	385	458	499	535	501	562	690	333	590	399
558	828	330	575	880	332	481	215	719	576	365	200	356	419	812	452	515	318
340	775	438	387	120	652	104	289	121	0	246	373	183	340	745	472	237	528
656	593	364	659	332	649	455	349	846	202	279	490	685	542	157	525	144	383
778	174	289	386	585	741	618	132	431	426	395	434	630	505	0	788	208	456
488	724	738	704	765	893	558	793	624	761	1031	555	778	1083	535	552	188	922
779	473	425	427	622	1015	655	718	343	565	978	663	487	138	855	307	284	138
235	571	0	446	162	111	268	624	559	362	543	671	458	451	524	419	736	455
436	861	217	207	279	700	557	128	325	82	398	793	313	376	175	465	756	563
142	220	513	187	223	391	289	226	360	0	166	437	247	404	749	435	150	590
556	597	402	663	295	612	459	387	827	189	343	554	666	531	221	589	208	372
759	137	177	450	589	722	675	196	495	389	459	498	694	569	80	635	290	0
523	81	188	255	596	636	439	620	648	430	528	496	496	813	427	513	938	294
284	193	777	634	205	297	159	475	870	390	453	119	437	833	535	219	139	590
168	131	310	208	303	279	92	367	0	235	371	187	344	581	348	151	364	469
429	240	495	208	525	291	225	682	32	283	488	521	378	103	384	150	219	614
94	165	384	421	577	454	92	429	302	254	432	489	364	165	569	224	154	301
0	369	205	289	446	537	328	286	355	483	371	375	437	343	554	269	360	673
116	385	322	512	369	149	238	230	209	605	237	300	352	378	568	445	172	281
436	108	332	343	218	290	421	164	354	201	149	0	121	570	386	543	518	142
84	297	263	570	35	636	29	319	432	36	534	236	482	687	373	238	301	581
349	222	466	162	55	583	562	429	381	294	625	96	452	631	687	562	336	765
423	299	500	212	347	0												

Table 7.5

Data of the 666-city-problem

No.	City	Coordinates		No.	City	Coordinates	
1	North Pole	90.00	0.00	7	Edmonton	53.33	−113.28
2	Barrow	71.17	−156.47	8	Calgary	51.03	−114.05
3	Fairbanks	64.51	−147.43	9	Regina	50.25	−104.39
4	Anchorage	61.13	−149.53	10	Saskatoon	52.07	−106.38
5	Juneau	58.20	−134.27	11	Winnipeg	49.53	−97.09
6	Vancouver	49.16	−123.07	12	Churchill, Can	58.46	−94.10

Table 7.5—continued

No.	City	Coordinates		No.	City	Coordinates	
13	Toronto	43.39	−79.23	64	Villahermosa	17.59	−92.55
14	Ottawa	45.25	−75.42	65	Merida	20.58	−89.37
15	Montreal	45.31	−73.34	66	Belize	17.30	−88.12
16	Quebec	46.49	−71.14	67	Guatemala City	14.38	−90.31
17	Halifax, Can	44.39	−63.36	68	San Salvador	13.42	−89.12
18	St. John's, Newf	47.34	−52.43	69	Tegucigalpa	14.06	−87.13
19	Seattle	47.36	−122.20	70	Managua	12.09	−86.17
20	Spokane	47.40	−117.23	71	San Jose	9.56	−84.05
21	Sacramento	38.35	−121.30	72	Panama	8.58	−79.32
22	San Francisco	37.48	−122.24	73	La Habana	23.08	−82.22
23	Los Angeles	34.03	−118.15	74	Santa Clara	22.24	−79.58
24	San Diego	32.43	−117.09	75	Santiago de Cuba	20.01	−75.49
25	Salt Lake City	40.46	−111.53	76	Kingston, Jam	18.00	−76.48
26	Phoenix, Ariz	33.27	−112.05	77	Port-au-Prince	18.32	−72.20
27	Denver, Colo	39.43	−105.01	78	Santo Domingo	18.28	−69.54
28	Albuquerque	35.05	−106.40	79	San Juan, P Rico	18.28	−66.07
29	El Paso, Tex	31.45	−106.29	80	Port-de-France	14.36	−61.05
30	Duluth, Minn	46.47	−92.06	81	Bridgetown	13.06	−59.37
31	Minneapolis	44.59	−93.13	82	Port of Spain	10.39	−61.31
32	Omaha	41.16	−95.57	83	Willemstad	12.06	−68.56
33	Kansas City	39.07	−94.39	84	Cayenne	4.56	−52.20
34	Oklahoma City	35.28	−97.32	85	Paramaribo	5.50	−55.10
35	Dallas	32.47	−96.48	86	Georgetown, Guy	6.48	−58.10
36	Houston, Tex	29.46	−95.22	87	Caracas	10.30	−66.56
37	Milwaukee	43.02	−87.55	88	Maracaibo	10.40	−71.37
38	Chicago	41.53	−87.38	89	Barranquilla	10.59	−74.48
39	St. Louis	38.39	−90.25	90	Medellin	6.15	−75.35
40	Memphis	35.08	−90.03	91	Bogota	4.36	−74.05
41	New Orleans	29.58	−90.07	92	Cali	3.27	−76.31
42	Detroit, Mich	42.20	−83.03	93	Villamil, Galap	−0.56	−91.01
43	Pittsburgh	40.26	−80.00	94	Quito	−0.13	−78.30
44	Cincinnati	39.06	−84.31	95	Riobamba	−1.40	−78.38
45	Atlanta, Ga	33.45	−84.23	96	Guayaquil	−2.10	−79.50
46	Boston, Mass	42.21	−71.04	97	Iquitos	−3.46	−73.15
47	New York	40.43	−74.01	98	Trujillo	−8.07	−79.02
48	Philadelphia	39.57	−75.07	99	Lima, Peru	−12.03	−77.03
49	Washington	38.54	−77.01	100	Cuzco	−13.31	−71.59
50	Jacksonville, Fl	30.20	−81.40	101	Arequipa	−16.24	−71.33
51	Miami	25.46	−80.12	102	La Paz, Bol	−16.30	−68.09
52	Nassau, Bahamas	25.05	−77.21	103	Santa Cruz	−17.48	−63.10
53	Chihuahua	28.38	−106.05	104	Potosi	−19.35	−65.45
54	Torreon	25.33	−103.26	105	Antofagasta	−23.39	−70.24
55	Monterrey	25.40	−100.19	106	Santiago de Chi	−33.27	−70.40
56	Tampico	22.13	−97.51	107	Concepion	−36.50	−73.03
57	San Luis Potosi	22.09	−100.59	108	Punta Arenas	−53.09	−70.55
58	Guadalajara	20.40	−103.20	109	Stanley, Falkl	−51.42	−57.51
59	Ciud. de Mexico	19.24	−99.09	110	Bahia Blanca	−38.43	−62.17
60	Puebla	19.03	−98.12	111	Mar Del Plata	−38.00	−57.33
61	Veracruz	19.20	−96.40	112	Montevideo	−34.50	−56.12
62	Acapulco	16.51	−99.55	113	Buenos Aires	−34.36	−58.27
63	Oaxaca	17.03	−96.43	114	Rosario, Arg	−32.57	−60.40

Table 7.5—continued

No.	City	Coordinates		No.	City	Coordinates	
115	Cordoba, Arg	−31.24	−64.11	166	Djibouti	11.36	43.09
116	Mendoza	−32.53	−68.49	167	Nouakchott	18.06	−15.57
117	Tucuman	−26.49	−65.13	168	Dakar	14.40	−17.26
118	Asuncion	−25.16	−57.40	169	Banjul	13.28	−16.39
119	Porto Alegre	−30.04	−51.11	170	Bissau	11.51	−15.35
120	Florianopolis	−27.35	−48.34	171	Tombouctou	16.46	−3.01
121	Curitiba	−25.25	−49.15	172	Bamako	12.39	−8.00
122	Sao Paulo	−23.32	−46.37	173	Kankan	10.23	−9.18
123	Rio de Janeiro	−22.54	−43.14	174	Conakry	9.31	−13.43
124	Ouro Preto	−20.23	−43.30	175	Freetown	8.30	−13.15
125	Belo Horizonte	−19.55	−43.56	176	Monrovia	6.18	−10.47
126	Campto Grande	−20.27	−54.37	177	Abidjan	5.19	−4.02
127	Cuiaba	−15.35	−56.05	178	Kumasi	6.41	−1.35
128	Goiania	−16.40	−49.16	179	Accra	5.33	−0.13
129	Brasilia	−15.47	−47.55	180	Lome	6.08	1.13
130	Salvador	−12.59	−38.31	181	PortoNovo, Ben	6.29	2.37
131	Recife	−8.03	−34.54	182	Ouagadougou	12.22	−1.31
132	Natal	−5.47	−35.13	183	Niamey	13.31	2.07
133	Fortaleza	−3.43	−38.30	184	Kano	12.00	8.30
134	Teresina	−5.05	−42.49	185	Maiduguri	11.51	13.10
135	Sao Luis	−2.31	−44.16	186	Ndjamena	12.07	15.03
136	Belem	−1.27	−48.29	187	Lagos	6.27	3.24
137	Manaus	−3.08	−60.01	188	Enugu	6.27	7.27
138	Porto Velho	−8.46	−63.54	189	Sao Tome	0.20	6.44
139	Praia, Cp Verfe	14.55	−23.31	190	Malabo	3.45	8.47
140	Las Palmas, Can	28.06	−15.24	191	Yaounde	3.52	11.31
141	Funchal, Madeira	32.38	−16.54	192	Bangui	4.22	18.35
142	Marrakech	31.38	−8.00	193	Libreville	0.23	9.27
143	Casablanca	33.39	−7.35	194	Brazzaville	−4.16	15.17
144	Rabat	34.02	−6.51	195	Kinshasa	−4.18	15.18
145	Fes	34.05	−4.57	196	Mbandaka	0.04	18.16
146	Tanger	35.48	−5.45	197	Kananga	−5.54	22.25
147	Oran	35.43	−0.43	198	Kisangani	0.30	25.12
148	Alger	36.47	3.03	199	Bujumbura	−3.23	29.22
149	Tamanrasset	22.56	5.30	200	Kigali	−1.57	30.04
150	Constantine	36.22	6.37	201	Kampala	0.19	32.25
151	Tunis	36.48	10.11	202	Nairobi	−1.17	36.49
152	Sfax	34.44	10.46	203	Mogadisho	2.01	45.20
153	Tarabulus	32.54	13.11	204	Mombasa	−4.03	39.40
154	Banghazi	32.07	20.04	205	Zanzibar	−6.10	39.11
155	Al-Iskandariyah	31.12	29.54	206	Dar-Es-Salaam	−6.48	39.17
156	Bur Said	31.16	32.18	207	Luanda	−8.48	13.14
157	As-Suways	29.58	32.33	208	Huambo	−12.44	15.47
158	Al-Qahirah	30.03	31.15	209	Lubumbashi	−11.40	27.28
159	Aswan	24.05	32.53	210	Kitwe	−12.49	28.13
160	Bur Sudan	19.37	37.14	211	Lusaka	−15.25	28.17
161	Al-Khurtum	15.36	32.32	212	Bulawayo	−20.09	28.36
162	Al-Ubayyid	13.11	30.13	213	Salisbury	−17.50	31.03
163	Al-Fashir	13.38	25.21	214	Blantyre	−15.47	35.00
164	Asmera	15.20	38.53	215	Beira, Moc	−19.49	34.52
165	Addis Abeba	9.00	38.50	216	Maputo	−25.58	32.35

Table 7.5—continued

No.	City	Coordinates		No.	City	Coordinates	
217	Saint Helena	−15.57	−5.42	268	Le Havre	49.30	0.08
218	Tristan Da Gunha	−37.15	−12.30	269	Paris	48.52	2.20
219	Walvisbaai	−22.59	14.31	270	Reims	49.15	4.02
220	Windhoek	−22.34	17.06	271	Dijon	47.19	5.01
221	Luederitz	−26.38	15.10	272	Nancy	48.41	6.12
222	Gaborone	−24.45	25.55	273	Strasbourg	48.35	7.45
223	Pretoria	−25.45	28.10	274	Luxembourg	49.36	6.09
224	Johannesburg	−26.15	28.00	275	Liege	50.38	5.34
225	Bloemfontein	−29.12	26.07	276	Bruxelles	50.50	4.20
226	Durban	−29.55	30.56	277	Lille	50.38	3.04
227	East London	−33.00	27.55	278	Gent	51.03	3.43
228	Port Elizabeth	−33.58	25.40	279	Antwerpen	51.13	4.25
229	Cape Town	−33.55	18.22	280	Eindhoven	51.26	5.28
230	Tulear	−23.21	43.40	281	Rotterdam	51.55	4.28
231	Antananarivo	−18.55	47.31	282	Amsterdam	52.22	4.54
232	Diego-Suarez	−12.16	49.17	283	Utrecht	52.05	5.08
233	Pt. Louis, Maur	−20.10	57.30	284	Groningen	53.13	6.33
234	Victoria, Seych	−4.38	55.27	285	Plymouth	50.23	−4.10
235	Pt. Delgada, Azr	37.44	−25.40	286	Bournemouth	50.43	−1.54
236	Lisboa	38.43	−9.08	287	Brighton	50.50	−0.08
237	Porto	41.11	−8.36	288	Cardiff	51.29	−3.13
238	Sevilla	37.23	−5.59	289	Bristol	51.27	−2.35
239	Cadiz	36.32	−6.18	290	London	51.30	−0.10
240	Malaga	36.43	−4.25	291	Birmingham	52.30	−1.50
241	Granada	37.13	−3.41	292	Liverpool	53.25	−2.55
242	Cordoba, Esp	37.53	−4.46	292	Manchester	53.30	−2.15
243	Alicante	38.21	−0.29	294	Sheffield	53.23	−1.30
244	Valencia	39.28	−0.22	295	Leeds	53.50	−1.35
245	Barcelona	41.23	2.11	296	Newcastle Up. T	54.59	−1.35
246	Zaragoza	41.38	−0.53	297	Edinburgh	55.57	−3.13
247	Madrid	40.24	−3.41	298	Glasgow	55.53	−4.15
248	Valladolid	41.39	−4.43	299	Dundee	56.28	−3.00
249	Bilbao	43.15	−2.58	300	Aberdeen	57.10	−2.04
250	La Coruna	43.22	−8.23	301	Lerwick, Shetl	60.09	−1.09
251	Ibiza	38.54	1.26	302	Torshavn, Faeroe	62.01	−6.46
252	Palma de Mallor	39.34	2.39	303	Cork	51.54	−8.28
253	Andorra	42.30	1.31	304	Limerick	52.40	−8.38
254	Bordeaux	44.50	−0.34	305	Dublin	53.20	−6.15
255	Toulouse	43.36	1.26	306	Belfast	54.35	−5.55
256	Marseille	43.18	5.24	307	Londonderry	55.00	−7.19
257	Nice	43.42	7.15	308	Reykjavik	64.09	−21.51
258	Monaco	43.42	7.23	309	Godthab	64.11	−51.44
259	Bastia, Fr	42.42	9.27	310	Thule	76.34	−68.47
260	Limoges	45.50	1.16	311	Hammerfest	70.40	23.42
261	St. Etienne	45.26	4.24	312	Narvik	68.26	17.25
262	Lyon	45.45	4.51	313	Oulu	65.01	25.28
263	Grenoble	45.10	5.43	314	Tampere	61.30	23.45
264	Brest	48.24	−4.29	315	Turku	60.27	22.17
265	Rennes	48.05	−1.41	316	Helsinki	60.10	24.58
266	Nantes	47.13	−1.33	317	Trondheim	63.25	10.25
267	Tours	47.23	0.41	318	Bergen	60.23	5.20

Table 7.5—continued

No.	City	Coordinates		No.	City	Coordinates	
319	Stavanger	58.58	5.45	370	Graz	47.05	15.27
320	Oslo	59.55	10.45	371	Torino	45.03	7.40
321	Goeteborg	57.43	11.58	372	Milano	45.28	9.12
322	Malmoe	55.36	13.00	373	Verona	45.27	11.00
323	Linkoeping	58.25	15.37	374	Venezia	45.27	12.21
324	Stockholm	59.20	18.03	375	Trieste	45.40	13.46
325	Visby	57.38	18.18	376	Genova	44.25	8.57
326	Arhus	56.09	10.13	377	Bologna	44.29	11.20
327	Odense	55.24	10.23	378	Firenze	43.46	11.15
328	Kobenhavn	55.40	12.35	379	San Marino	43.55	12.28
329	Bremen	53.04	8.49	380	Cagliari	39.20	9.00
330	Hamburg	53.33	9.59	381	Roma	41.54	12.29
331	Kiel	54.20	10.08	382	Napoli	40.51	14.17
332	Rostock	54.05	12.07	383	Foggia	41.27	15.34
333	Muenster	51.57	7.37	384	Bari	41.07	16.52
334	Hannover	52.24	9.44	385	Taranto	40.28	17.15
335	Magdeburg	52.07	11.38	386	Messina	38.11	15.33
336	Berlin	52.31	13.24	387	Catania	37.30	15.06
337	Aachen	50.47	6.05	388	Palermo	38.07	13.21
338	Bonn	50.44	7.05	389	Valetta, Malta	35.54	14.31
339	Koeln	50.56	6.59	390	Szczecin	53.24	14.32
340	Duesseldorf	51.12	6.47	391	Gdansk	54.23	18.40
341	Schwelm	51.17	7.17	392	Bydgoszcz	53.08	18.00
342	Essen	51.28	7.01	393	Poznan	52.25	16.55
343	Bochum	51.28	7.13	394	Lodz	51.46	19.30
344	Herne	51.32	7.13	395	Warszawa	52.15	21.00
345	Dortmund	51.31	7.28	396	Bialystok	53.09	23.09
346	Kassel	51.19	9.29	397	Wroclaw	51.06	17.00
347	Erfurt	50.58	11.01	398	Katowice	50.16	19.00
348	Halle	51.29	11.58	399	Krakow	50.03	19.58
349	Leipzig	51.19	12.20	400	Lublin	51.15	22.35
350	Karl-Marx-Stadt	50.50	12.55	401	Plzen	49.45	13.23
351	Dresden	51.03	13.44	402	Praha	50.05	14.26
352	Saarbruecken	49.14	6.59	403	Ostrava	49.50	18.17
353	Frankfurt/Main	50.07	8.40	404	Brno	49.12	16.37
354	Heidelberg	49.25	8.43	405	Bratislava	48.09	17.07
355	Wuerzburg	49.48	9.56	406	Kosice	48.43	21.15
356	Nuernberg	49.27	11.04	407	Budapest	47.30	19.05
357	Karlsruhe	49.03	8.24	408	Debrecen	47.32	21.38
358	Stuttgart	48.46	9.11	409	Pecs	46.05	18.13
359	Regensburg	49.01	12.06	410	Szeged	46.15	20.09
360	Muenchen	48.08	11.34	411	Timisoara	45.45	21.13
361	Geneve	46.12	6.09	412	Cluj	46.47	23.36
362	Lausanne	46.31	6.38	413	Iasi	47.10	27.35
363	Bern	46.57	7.26	414	Sibiu	45.48	24.09
364	Basel	47.33	7.35	415	Brasov	45.39	25.37
365	Zuerich	47.23	8.32	416	Bucuresti	44.26	26.06
366	Innsbruck	47.16	11.24	417	Constanta	44.11	28.39
367	Salzburg	47.48	13.02	418	Ljubljana	46.03	14.31
368	Linz	48.18	14.18	419	Rijeka	45.20	14.27
369	Wien	48.13	16.20	420	Zagreb	45.48	15.58

Table 7.5—continued

No.	City	Coordinates		No.	City	Coordinates	
421	Split	43.31	16.27	472	Samarkand	39.40	66.48
422	Sarajevo	43.52	18.25	473	Dusanbe	38.35	68.48
423	Beograd	44.50	20.30	474	Wulumuqi	43.48	87.35
424	Dubrovnik	42.38	18.07	475	Irkutsk	52.16	104.20
425	Skopje	41.59	21.26	476	Ulaan Baatar	47.55	106.53
426	Tirane	41.20	19.50	477	Cita	52.03	113.30
427	Sofija	42.41	23.19	478	Jakutsk	62.13	129.49
428	Plovdiv	42.09	24.45	479	Anadyr	64.45	177.29
429	Varna	43.13	27.55	480	Petropavlovsk	53.01	158.39
430	Burgas	42.30	27.28	481	Magadan	59.34	150.48
431	Kerkira	39.36	19.56	482	Blagovescensk	50.17	127.32
432	Thessaloniki	40.38	22.56	483	Komsomolsk	50.35	137.02
433	Patrai	38.15	21.44	484	Chabarovsk	48.27	135.06
434	Athinai	37.58	23.43	485	Juzno-Sachalinsk	46.58	142.42
435	Iraklion	35.20	25.09	486	Vladivostok	43.10	131.56
436	Levkosia	35.10	33.22	487	Istanbul	41.01	28.58
437	Murmansk	68.58	33.05	488	Izmir	38.25	27.09
438	Archangelsk	64.34	40.32	489	Ankara	39.56	32.52
439	Leningrad	59.55	30.15	490	Kayseri	38.43	35.30
440	Tallinn	59.25	24.45	491	Sivas	39.45	37.02
441	Riga	56.57	24.06	492	Erzurum	39.55	41.17
442	Kaliningrad	54.43	20.30	493	Diyarbakir	37.55	40.14
443	Vilnius	54.41	25.19	494	Adana	37.01	35.18
444	Minsk	53.54	27.34	495	Halab	36.12	37.10
445	Lvov	49.50	24.00	496	Hims	34.44	36.43
446	Kijev	50.26	30.31	497	Dimashq	33.30	36.18
447	Odessa	46.28	30.44	498	Bayrut	33.53	35.30
448	Moskva	55.45	37.35	499	Amman	31.57	35.56
449	Gorkij	56.20	44.00	500	Hefa	32.50	35.00
450	Kazan	55.45	49.08	501	Tel Aviv	32.04	34.46
451	Kujbysev	53.12	50.09	502	Yerushalayim	31.46	35.14
452	Voronez	51.40	39.10	503	Al-Madinah	24.28	39.36
453	Charkov	50.00	36.15	504	Juddah	21.30	39.12
454	Dnepropetrovsk	48.27	34.59	505	Makkah	21.27	39.49
455	Sevastopol	44.36	33.32	506	Sana	15.23	44.12
456	Rostov-na-donu	47.14	39.42	507	Al-Hudaydah	14.48	42.57
457	Volgograd	48.44	44.25	508	Aden	12.45	45.12
458	Astrachan	46.21	48.03	509	Al-Mukalla	14.32	49.08
459	Tbilisi	41.43	44.49	510	Masqat	23.37	58.35
460	Jerevan	40.11	44.30	511	Dubayy	25.18	55.18
461	Baku	40.23	49.51	512	Ad-Dawhah	25.17	51.32
462	Perm	58.00	56.15	513	Al-Manamah	26.13	50.35
463	Sverdlovsk	56.51	60.36	514	Ar-Riyad	24.38	46.43
464	Vorkuta	67.27	63.58	515	Al-Kuwayt	29.20	47.59
465	Norilsk	69.20	88.06	516	Al-Basrah	30.30	47.47
466	Omsk	55.00	73.24	517	Baghdad	33.21	44.25
467	Novosibirsk	55.02	82.55	518	Kirkuk	35.28	44.28
468	Krasnojarsk	56.01	92.50	519	Al-Mawsil	36.20	43.08
469	Karaganda	49.50	73.10	520	Tabriz	38.05	46.18
470	Alma Ata	43.15	76.57	521	Rasht	37.16	49.36
471	Taskent	41.20	69.18	522	Tehran	35.40	51.26

Table 7.5—continued

No.	City	Coordinates		No.	City	Coordinates	
523	Kermanshah	34.19	47.04	574	Bangkok	13.45	100.31
524	Abadan	30.20	48.16	575	Pinang	5.25	100.20
525	Esfahan	32.40	51.38	576	Kuala Lumpur	3.10	101.42
526	Shiraz	29.36	52.32	577	Singapore	1.17	103.51
527	Kerman	30.17	57.05	578	Medan	3.35	98.40
528	Mashhad	36.18	59.36	579	Padang	−0.57	100.21
529	Herat	34.20	62.12	580	Palembang	−2.55	104.45
530	Quandahar	31.32	65.30	581	Jakarta	−6.10	106.48
531	Kabul	34.31	69.12	582	Bandung	−6.54	107.36
532	Rawalpindi	33.36	73.04	583	Yogyakarta	−7.48	110.22
533	Lahore	31.35	74.18	584	Surabaya	−7.15	112.45
534	Lyallpur	31.25	73.05	585	Denpasar	−8.39	115.13
535	Multan	30.11	71.29	586	Kupang	−10.10	123.35
536	Quetta	30.12	67.00	587	Banjarmasin	−3.20	114.35
537	Sukkur	27.42	68.52	588	Kuching	1.33	110.20
538	Hyderabad, Pak	25.22	68.22	589	Brunei	4.56	114.55
539	Karachi	24.52	67.03	590	Samarinda	−0.30	117.09
540	Dehra Dun	30.19	78.02	591	Ujung Pandang	−5.07	119.24
541	Delhi	28.40	77.13	592	Manado	1.29	124.51
542	Jodhpur	26.17	73.02	593	Ambon	−3.43	128.12
543	Jaipur	26.55	75.49	594	Tual	−5.40	132.45
544	Kanpur	26.28	80.21	595	Davao	7.04	125.36
545	Varanasi	25.20	83.00	596	Cebu	10.18	123.54
546	Patna	25.36	85.07	597	Iloilo	10.42	122.34
547	Calcutta	22.32	88.22	598	Manila	14.35	121.00
548	Ahmadabad	23.02	72.37	599	Hongkong	22.17	114.09
549	Nagpur	21.09	79.06	600	Kaohsiung	22.38	120.17
550	Cuttack	20.30	85.50	601	Taipei	25.03	121.30
551	Bombay	18.58	72.50	602	Lasa	29.40	91.09
552	Hyderabad, India	17.23	78.29	603	Lanzhou	36.03	103.41
553	Vishakhapatnam	17.42	83.18	604	Xian	34.15	108.52
554	Hubli	15.21	75.10	605	Chengdu	30.39	104.04
555	Bangalore	12.59	77.35	606	Chongqing	29.39	106.34
556	Madras	13.05	80.17	607	Kunming	25.05	102.40
557	Tiruchchirappa.	10.49	78.41	608	Guangzhou	23.06	113.16
558	Madurai	9.56	78.07	609	Fuzhou	26.06	119.17
559	Colombo	6.56	79.51	610	Wuhan	30.36	114.17
560	Katmandu	27.43	85.19	611	Nanjing	32.03	118.47
561	Thimbu, Bhutan	27.28	89.39	612	Shanghai	31.14	121.28
562	Dacca	23.43	90.25	613	Zhengzhou	34.48	113.39
563	Chittagong	22.20	91.50	614	Qingdao	36.06	120.19
564	Mandalay	22.00	96.05	615	Taiyuan	37.55	112.30
565	Rangoon	16.47	96.10	616	Tianjin	39.08	117.12
566	Chiang Mai	18.47	98.59	617	Beijing	39.55	116.25
567	Luangphrabang	19.52	102.08	618	Luda(Dairen)	38.53	121.35
568	Viangchan	17.58	102.36	619	Shenyang	41.48	123.27
569	Hanoi	21.02	105.51	620	Haerbin	45.45	126.41
570	Hue	16.28	107.36	621	Pyongyang	39.01	125.45
571	Da-Nang	16.04	108.13	622	Soul	37.33	126.58
572	Saigon	10.45	106.40	623	Pusan	35.06	129.03
573	Phnum Penh	11.33	104.55	624	Sapporo	43.03	141.21

Table 7.5—continued

No.	City	Coordinates		No.	City	Coordinates	
625	Akita	39.43	140.07	646	Brisbane	−27.28	153.02
626	Sendai	38.15	140.53	647	Townsville	−19.16	146.48
627	Tokyo	35.42	139.46	648	Alice Springs	−23.42	133.53
628	Nagoya	35.10	136.55	649	Dunedin	−45.52	170.30
629	Kanazawa	36.34	136.39	650	Christchurch	−43.32	172.38
630	Kyoto	35.00	135.45	651	Wellington	−41.18	174.47
631	Osaka	34.40	135.30	652	Auckland	−36.52	174.46
632	Hiroshima	34.24	132.27	653	Nukualofa, Tonga	−21.08	−175.12
633	Nagasaki	32.48	129.55	654	Paga Pago, Samoa	−14.16	−170.42
634	Kagoshima	31.36	130.33	655	Suva, Fiji	−18.08	178.25
635	Naha, Okinawa	26.13	127.40	656	Noumea, N. Caled	−22.16	166.27
636	Guam	13.28	144.47	657	Honiara, Solomon	−9.26	159.57
637	Jayapura, Irian	−2.32	140.42	658	Nauru	−0.32	166.55
638	Rabaul	−4.12	152.12	659	Bikini	11.35	165.23
639	Port Moresby	−9.30	147.10	660	Honululu	21.19	−157.52
640	Darwin	−12.28	130.50	661	Christmas Isl.	1.52	−157.20
641	Perth	−31.56	115.50	662	Hiva Oa, Marques	−9.45	−139.00
642	Adelaide	−34.55	138.35	663	Papeete, Tahiti	−17.32	−149.34
643	Melbourne	−37.49	144.58	664	Pitcairn	−25.04	−130.06
644	Hobart	−42.53	147.19	665	Pascua, Isla de	−27.07	−109.22
645	Sydney	−33.52	151.13	666	South Pole	−90.00	0.00

Table 7.6

Conversion routine for the 666-city-problem

```
program comdist(input,output);
const pi = 3.141592;
      r  = 6378.388;                          {equator radius of the earth}
      max_n =1000;                            {maximum number of points}
var n,                                        {number of points}
    lauf,
    i,j      : integer;
    ll,bb    : real;
    b,                                        {geographic latitude in rad}
    l                                         {geographic longitude in rad}
             : array(.1..max_n.) of real;
    d                                         {distance table}
             : array(.1..max_n,1..max_n.) of real;

 function acos ( x:real) : real;              {compute arcus cosine}
 begin
  acos := pi/2.0 - arctan(x/sqrt(1.0 - x * x));
 end;

 function radian(degrees : real) : real;      {convert degrees into radian}
 var deg,min : real;
 begin
  deg := trunc(degrees);
  min := degrees - deg;
  radian := pi * ( deg + 5.0 * min / 3.0) / 180.0;
 end;

 function dist(b1,l1,b2,l2 : real) : real;    {compute distance between}
 var cbdiff,cldiff,cbsum : real;              {two points}
 begin
  cldiff := cos(l1 - l2);
  cbdiff := cos(b1 - b2);
  cbsum  := cos(b1 + b2);
  dist   := r * acos( 0.5 * (( 1.0 + cldiff ) * cbdiff
                           -( 1.0 - cldiff ) * cbsum )) + 1.0;
 end;
```

Table 7.6—continued

```
begin  {main}
  reset(input);
  read(n);
  if (n < 1) or (n > max_n)
     then begin
            rewrite(output);
            writeln(' Number of points too large or not positive');
            halt;
          end;
  for i:= 1 to n do begin        {input of coordinates}
    read(bb,ll);
    b(.i.) := radian(bb);
    l(.i.) := radian(ll);
  end;

  for i:= 1 to n do
  begin
    d(.i,i.) := 0;
  end;

  for i:= 1 to n - 1 do          {distance calculation}
    for j:= i + 1 to n do
      d(.i,j.) := dist( b(.i.),l(.i.),b(.j.),l(.j.) );

  rewrite(output);

  writeln('10');
  writeln(' ',n:6);
  writeln('(12 I 6 )');
  lauf := 0 ;
  for j:= 1 to n do
  begin   {output of distance table}
    for i:= 1 to j do
    begin
      if lauf = 12 then
      begin
        writeln ;
        lauf := 0 ;
      end ;
      write(trunc(d(.i,j.)):6);
      lauf := lauf + 1 ;
    end ;
  end;
  writeln ;
end.
```

Table 7.7

Coordinates for the 442-PCB-problem

20	40	20	50	20	60	20	70	20	80	20	90	20	100
20	110	20	120	20	130	20	140	20	150	20	160	20	170
20	180	20	190	20	200	20	210	20	220	20	230	20	240
20	250	20	260	20	270	20	280	20	290	20	300	20	310
20	320	20	330	20	340	20	350	20	360	30	40	30	50
30	60	30	70	30	80	30	90	30	100	30	110	30	120
30	130	30	140	30	150	30	160	30	170	30	180	30	190
30	200	30	210	30	220	30	230	30	240	30	250	30	260
30	270	30	280	30	290	30	300	30	310	30	320	30	330
30	340	30	350	40	40	40	50	40	60	40	70	40	80
40	90	40	100	40	110	40	120	40	130	40	140	40	150
40	160	40	170	40	180	40	190	40	200	40	210	40	220
40	230	40	240	40	250	40	260	40	270	40	280	40	290

Table 7.7—continued

40	300	40	310	40	320	40	330	40	340	40	350	40	360
50	150	50	183	50	310	60	40	70	30	70	60	70	150
70	160	70	180	70	210	70	240	70	270	70	300	70	330
70	360	80	30	80	60	80	103	80	150	80	180	80	210
80	240	80	260	80	270	80	300	80	330	80	360	90	30
90	60	90	150	90	180	90	210	90	240	90	270	90	300
90	330	90	360	100	30	100	60	100	110	100	150	100	163
100	180	100	210	100	240	100	260	100	270	100	300	100	330
100	360	110	30	110	60	110	70	110	90	110	150	110	180
110	210	110	240	110	270	110	300	110	330	110	360	120	30
120	60	120	150	120	170	120	180	120	210	120	240	120	270
120	300	120	330	120	360	130	30	130	60	130	70	130	113
130	150	130	180	130	210	130	220	130	240	130	270	130	300
130	330	130	360	140	30	140	60	140	93	140	150	140	180
140	200	140	210	140	240	140	250	140	270	140	282	140	290
140	300	140	330	140	360	150	150	150	180	150	190	150	210
150	240	150	270	150	280	150	286	150	300	150	330	150	360
160	110	160	130	160	150	160	180	160	210	160	240	160	270
160	300	160	330	160	360	170	120	170	150	170	180	170	210
170	240	170	360	180	30	180	60	180	123	180	150	180	180
180	210	180	240	190	30	190	60	190	300	190	352	200	30
200	37	200	60	200	80	200	90	200	100	200	110	200	120
200	130	200	140	200	150	200	160	200	170	200	180	200	190
200	200	200	210	200	220	200	230	200	240	200	250	200	260
200	270	200	280	200	290	200	300	200	310	200	350	210	30
210	60	210	320	220	30	220	47	220	60	220	320	230	30
230	60	230	340	240	30	240	60	240	210	250	30	250	80
260	40	260	50	260	80	260	90	260	100	260	110	260	120
260	130	260	140	260	150	260	160	260	170	260	180	260	190
260	200	260	210	260	220	260	230	260	240	260	250	260	260
260	270	260	280	260	290	260	300	260	310	260	340	270	70
270	80	270	90	270	100	270	110	270	120	270	130	270	140
270	150	270	160	270	170	270	180	270	190	270	200	270	210
270	220	270	230	270	250	270	260	270	270	270	280	270	290
270	300	270	310	270	320	270	330	270	340	270	350	270	360
270	370	270	380	280	90	280	113	290	40	290	50	290	140
290	240	290	300	300	70	300	80	300	90	300	100	300	110
300	120	300	130	300	150	300	160	300	170	300	180	300	190
300	200	300	210	300	220	300	230	300	250	300	260	300	270
300	280	300	290	300	300	300	310	300	320	300	330	300	340
300	350	300	360	300	370	300	380	15	350	15	355	47	255
47	335	47	345	54	233	54	243	62	365	62	371	75	255
85	52	85	70	85	228	94	74	95	222	91	260	105	105
115	135	117	228	122	221	135	75	135	170	135	214	145	77
155	30	155	50	155	185	165	105	169	268	171	31	171	51
175	75	179	258	172	261	179	333	172	341	183	270	183	280
183	345	206	165	205	315	217	190	211	200	212	275	215	325
229	140	222	282	228	325	239	130	232	150	245	71	262	365
275	52	276	236	285	220	285	270	285	335	293	95	295	175
295	205	52	320	230	350	232	315	53	210	255	71	75	49
0	0												

The optimal solutions are given in Tables 7.8, ..., 7.18 and Pictures 7.1, ..., 7.7. The tours are defined by the sequence of cities to be visited which is given in a rowwise fashion.

Table 7.8

Optimal 17-city-tour, length: 2085

16	12	9	5	2	10	11	3	15	14	17	6	8	7	13	4	1

Table 7.9

Optimal 21-city-tour, length: 2707

7	8	6	16	5	9	3	2	21	15	14	13	18	10	17	19	20	11	4	12	1

Table 7.10

Optimal 24-city-tour, length: 1272

16	11	3	7	6	24	8	21	5	10	17	22	18	19	15	2	20	14	13	9	23	4	12	1

Table 7.11

Optimal 48-city-tour, length: 5046

29	7	28	44	41	46	18	34	23	25	3	19	4	30	38	20
35	42	39	40	2	45	43	47	37	24	15	10	12	31	5	33
8	22	21	17	27	32	9	14	6	2	36	11	16	48	13	1

Table 7.12

Optimal Africa-Tour (96 cities), length: 55 209

29	2	3	4	5	6	7	8	9	10	12	13	14	15	16	17
20	18	19	21	25	24	23	22	26	28	27	65	96	94	95	93
92	77	76	68	67	66	64	63	62	61	60	59	71	72	73	75
74	84	86	85	78	88	87	89	90	91	83	82	81	80	79	70
69	57	56	58	54	48	47	46	50	52	53	55	51	49	43	42
41	40	39	44	45	11	33	34	35	38	37	36	32	31	30	1

Table 7.13

Optimal America-Tour (137 cities), length: 69 853

2	3	4	5	18	19	7	6	9	8	26	24	20	21	22	23
25	27	28	52	53	54	55	56	57	61	58	59	60	62	63	64
65	66	67	68	69	70	92	97	98	99	100	101	104	103	102	126
125	117	116	114	115	105	106	107	108	109	110	111	112	113	118	119
120	121	122	123	124	127	128	129	130	131	132	133	134	135	83	84
85	136	137	96	94	95	93	91	90	89	71	88	87	82	86	81
80	79	78	77	76	75	74	51	73	72	50	49	44	43	42	48
47	46	45	16	17	15	14	13	12	41	36	37	38	39	40	35
34	33	32	31	30	29	10	11	1							

Table 7.14

Optimal Europe-tour (202 cities), length: 40 160

1	3	16	14	13	12	15	20	26	33	32	31	30	51	69	70
71	72	73	64	63	65	66	62	61	60	59	58	57	54	55	52
56	53	34	35	36	40	118	38	37	28	27	29	127	128	129	131
130	39	123	124	120	119	121	122	125	126	132	133	134	167	168	117
116	115	114	113	112	99	111	110	109	108	107	106	105	104	103	41
42	43	44	45	46	47	49	48	50	95	100	101	102	156	98	96
97	93	92	94	88	87	86	83	84	85	67	68	74	75	76	77
78	79	80	82	81	90	89	91	157	158	159	163	160	161	162	166
165	164	169	170	135	171	173	172	174	178	180	181	179	182	183	195
196	202	201	200	199	197	198	194	193	191	192	190	187	188	189	177
176	175	186	136	184	185	141	140	139	143	144	145	147	148	149	150
151	152	153	155	154	146	25	142	138	137	24	23	22	21	19	11
18	17	10	9	7	6	8	4	5	2						

Table 7.15

Optimal Asia-tour (229 cities), length: 134 602

1	28	29	31	32	39	40	41	42	45	44	43	224	225	226	229
228	227	218	217	219	220	216	215	214	213	208	207	206	209	210	211
203	202	221	222	223	200	201	158	204	212	205	149	148	147	146	145
144	143	142	139	140	141	152	153	154	151	155	150	157	156	159	160
161	162	164	165	173	163	172	174	175	176	199	198	197	187	196	195
194	192	193	191	190	189	188	49	47	48	46	184	50	186	185	183
182	178	180	181	179	177	168	167	169	170	171	133	134	135	136	137
138	132	131	130	129	128	127	126	111	114	117	113	116	120	123	122
121	119	118	115	112	103	102	101	100	94	93	92	36	35	37	95
96	97	98	99	106	107	105	104	108	109	110	124	125	166	38	34
33	30	27	26	15	14	13	12	16	17	18	20	21	22	23	24
56	57	83	82	81	87	84	25	85	86	89	88	80	79	90	91
74	75	76	77	78	73	72	70	71	69	68	67	63	66	65	64
62	61	60	59	58	54	55	53	52	51	19	11	10	9	8	7
6	5	4	3	2											

Table 7.16

Optimal Australia-Asia-Europe-tour (431 cities), length: 171 414

1	2	5	4	8	6	7	9	10	17	18	11	19	21	22	23
24	137	138	142	25	146	154	155	153	152	151	150	149	148	147	145
144	143	139	140	141	185	184	186	187	190	188	189	177	176	175	173
174	178	180	181	182	183	195	196	194	193	198	191	192	197	199	200
201	254	253	255	221	213	179	212	220	219	218	222	223	224	225	226
286	227	287	288	291	292	281	290	282	289	283	284	285	259	258	257
256	260	261	262	263	264	202	266	267	268	265	269	270	271	273	272
274	275	280	279	278	277	276	293	294	295	296	302	305	304	303	301
300	299	298	297	239	238	237	236	240	368	327	326	312	311	310	306
307	309	308	314	317	320	321	323	324	325	322	318	315	319	316	313
328	329	330	331	332	333	334	340	339	338	337	336	335	373	372	371
369	370	379	381	383	382	384	385	387	388	380	378	377	376	374	365
375	367	366	364	363	362	361	358	359	360	406	352	357	353	356	355
354	343	342	341	344	345	346	347	348	349	350	351	407	414	413	412
411	408	409	410	415	416	417	418	422	421	419	420	429	430	431	428
427	426	425	424	423	404	405	403	402	401	400	399	389	398	397	396
394	395	393	392	391	390	251	249	250	252	386	248	243	242	241	234
233	235	232	229	228	217	216	215	214	205	206	82	81	80	79	78
77	203	204	230	231	244	247	246	245	76	75	74	68	67	85	84
83	86	87	89	90	91	207	209	210	162	208	157	158	159	163	160
161	166	211	172	165	164	169	170	171	135	136	134	133	132	126	125
167	168	117	116	115	114	101	102	156	98	88	94	92	93	97	96
95	100	112	113	122	121	119	120	124	123	39	130	131	129	128	127
29	27	28	37	38	118	40	41	103	104	105	106	107	108	109	110
111	99	50	48	49	47	46	45	42	44	43	36	35	34	53	56
52	55	54	57	58	59	60	61	62	66	65	63	64	73	72	71
70	69	51	30	31	32	33	26	20	15	12	13	14	16	3	

Table 7.17

Optimal tour of the PCB-problem (442 nodes), length: 5069

1	2	3	4	5	6	7	8	41	42	9	10	43	44	11	12
13	14	15	16	17	18	51	52	19	20	53	85	381	382	86	54
21	22	23	56	55	87	378	88	89	90	91	92	60	59	58	57
24	25	26	27	28	61	93	101	111	123	133	146	158	169	182	197
196	195	194	181	168	157	145	144	391	132	122	110	121	385	109	120
388	131	143	156	167	180	192	193	204	216	225	233	408	409	412	413
404	217	205	206	207	208	218	219	209	198	183	170	159	147	134	124
112	436	94	62	29	30	31	32	376	377	33	65	64	63	95	379
96	380	97	98	384	383	113	125	135	148	160	171	184	199	210	220
226	411	410	414	237	265	437	275	423	438	272	420	268	416	264	263
236	262	261	422	419	260	259	258	257	256	255	254	253	418	417	252
251	250	415	249	248	247	246	245	244	243	242	241	407	228	235	240
267	271	270	274	277	426	280	440	308	309	283	284	310	339	311	285
286	312	340	313	287	288	314	315	289	424	421	425	290	316	317	291
292	318	319	293	294	320	321	295	278	297	296	322	323	430	429	324
298	299	300	325	326	301	302	327	431	328	303	304	329	330	305	306
331	332	333	432	334	307	335	336	427	337	338	375	374	373	372	371
370	369	368	345	367	366	365	364	363	362	344	361	360	359	435	358
357	356	434	355	354	353	343	352	351	350	349	433	348	347	346	342
341	428	282	281	279	276	273	269	266	238	239	234	227	405	406	401
400	185	172	161	149	136	126	114	103	102	441	104	115	386	127	387

Table 7.17—continued

389	116	138	392	152	151	137	150	162	173	186	174	396	399	187	175
211	403	229	221	212	230	222	213	200	188	176	163	393	153	139	140
128	117	105	106	107	118	129	141	154	165	164	397	177	189	201	202
402	214	223	231	232	224	215	203	190	191	398	178	179	394	395	166
155	142	390	130	119	108	439	84	83	82	50	49	81	100	80	48
47	79	78	46	45	77	99	76	75	74	73	72	40	39	71	70
38	37	69	68	36	35	67	66	34	442						

Table 7.18

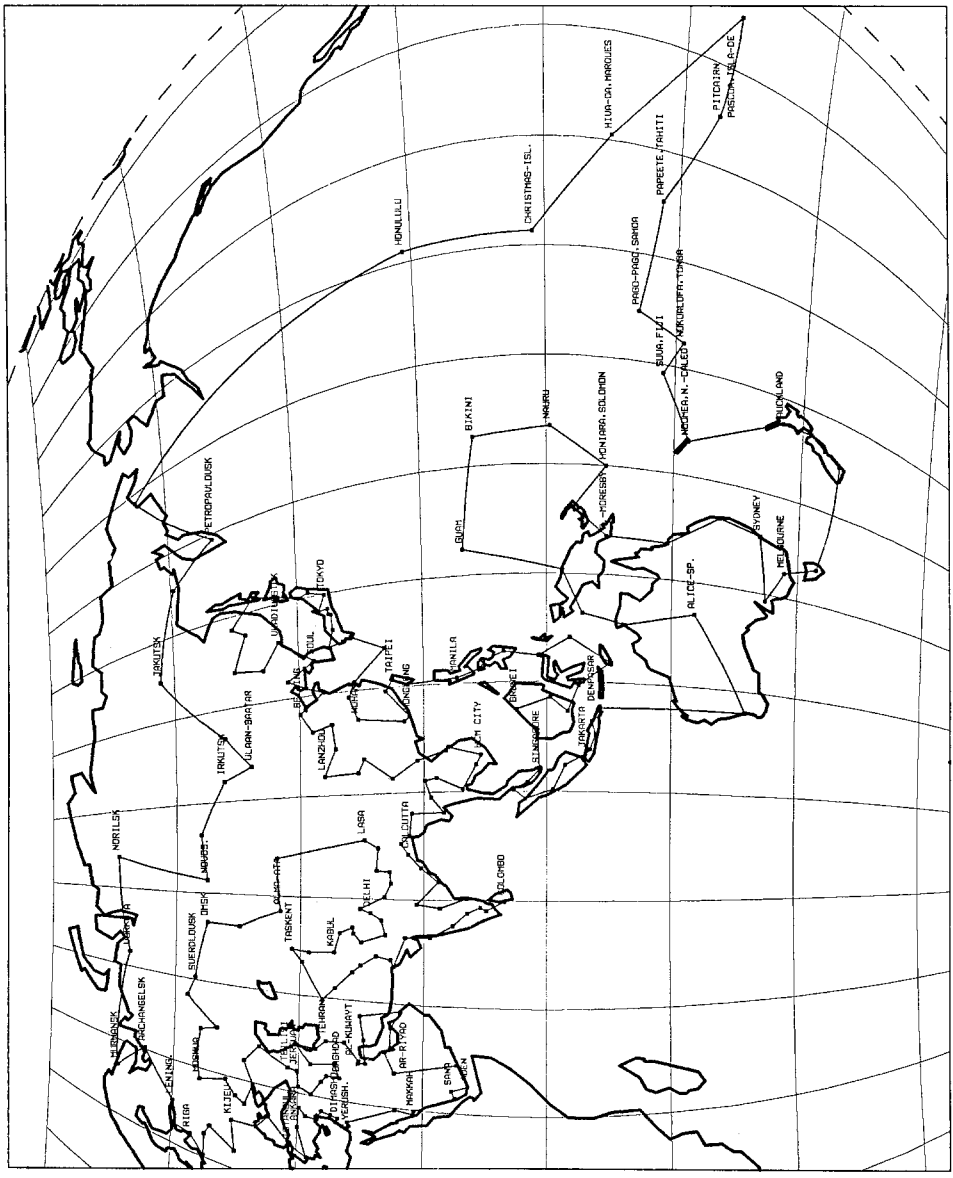
Optimal world-tour (666 cities), length: 294 358

1	465	464	463	462	451	450	449	448	452	453	454	456	457	458	459
460	520	461	521	522	525	526	515	524	516	523	517	518	519	493	492
491	490	494	495	496	497	498	436	500	499	502	501	156	155	158	157
159	161	162	163	186	185	184	149	183	182	171	172	173	167	139	168
169	170	174	175	176	177	178	179	180	181	187	188	190	189	193	191
192	196	194	195	207	208	217	218	219	220	221	229	228	227	225	226
216	223	224	222	212	213	211	210	209	197	198	199	200	201	202	204
205	206	214	215	230	231	233	232	234	203	165	166	508	509	506	507
164	160	504	505	503	514	513	512	511	510	527	528	529	530	536	537
538	539	548	551	554	555	557	558	559	556	552	549	553	550	547	562
563	564	566	565	574	568	567	569	570	571	572	573	575	578	576	577
579	580	581	582	583	584	585	587	588	589	590	591	586	640	648	641
642	643	644	645	646	647	639	638	657	658	659	636	637	594	593	592
595	596	597	598	600	601	609	599	608	610	611	612	635	634	633	623
632	631	630	628	629	627	626	625	624	485	483	484	482	620	486	622
621	619	618	614	616	617	615	613	604	603	605	606	607	602	561	560
546	545	544	540	541	543	542	535	534	533	532	531	473	472	471	470
474	469	466	467	468	475	476	477	478	481	480	479	2	3	4	5
6	19	20	8	7	10	9	12	11	30	31	32	33	39	40	41
36	35	34	27	25	21	22	23	24	26	28	29	53	54	55	56
57	58	62	59	60	61	63	64	65	66	67	68	69	70	71	93
665	664	662	663	661	660	654	653	655	656	652	651	650	649	666	108
109	111	110	107	106	116	115	114	113	112	119	120	121	122	123	124
125	128	129	130	131	132	133	134	135	136	84	85	86	137	138	127
126	118	117	105	104	103	102	101	100	99	98	97	95	96	94	92
91	90	72	89	88	83	87	82	81	80	79	78	77	76	75	52
74	73	51	50	45	44	38	37	42	13	43	49	48	47	46	14
15	16	17	18	235	141	140	142	143	144	145	146	239	240	241	242
238	236	237	250	248	247	246	249	254	255	253	245	252	251	244	243
147	148	150	380	259	376	372	371	258	257	256	263	262	261	260	267
266	265	264	285	303	304	305	306	307	298	297	299	300	296	295	294
293	292	291	288	289	286	290	287	268	269	270	277	278	276	279	280
281	283	282	284	333	345	344	343	342	341	340	339	338	337	275	274
352	272	271	361	362	363	365	364	273	357	358	354	353	355	356	347
346	334	329	330	331	327	326	328	322	332	390	336	335	348	349	350
351	402	401	359	360	366	367	368	370	420	418	419	375	374	373	377
378	379	381	382	383	384	385	386	387	389	388	151	152	153	154	435
488	434	433	431	432	428	427	425	426	424	421	422	423	411	410	409
407	405	369	404	403	398	399	394	397	393	392	391	442	396	395	400
445	406	408	412	414	415	416	417	429	430	487	489	455	447	413	446
444	443	441	440	316	439	438	437	311	312	313	314	315	324	325	323
321	320	317	318	319	301	302	308	309	310						

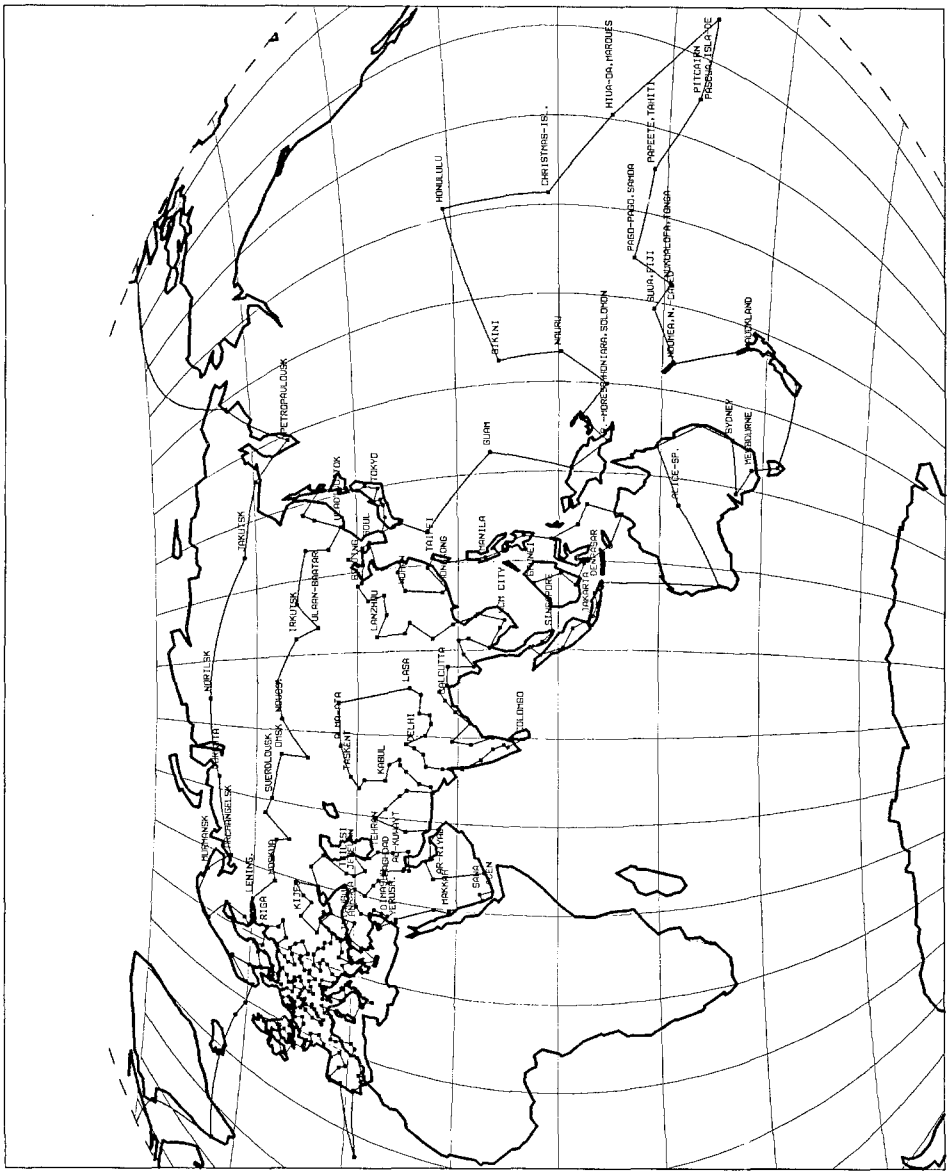
Picture 7.1. Optimal Africa-tour (96 cities).

Picture 7.2. Optimal America-tour (137 cities).

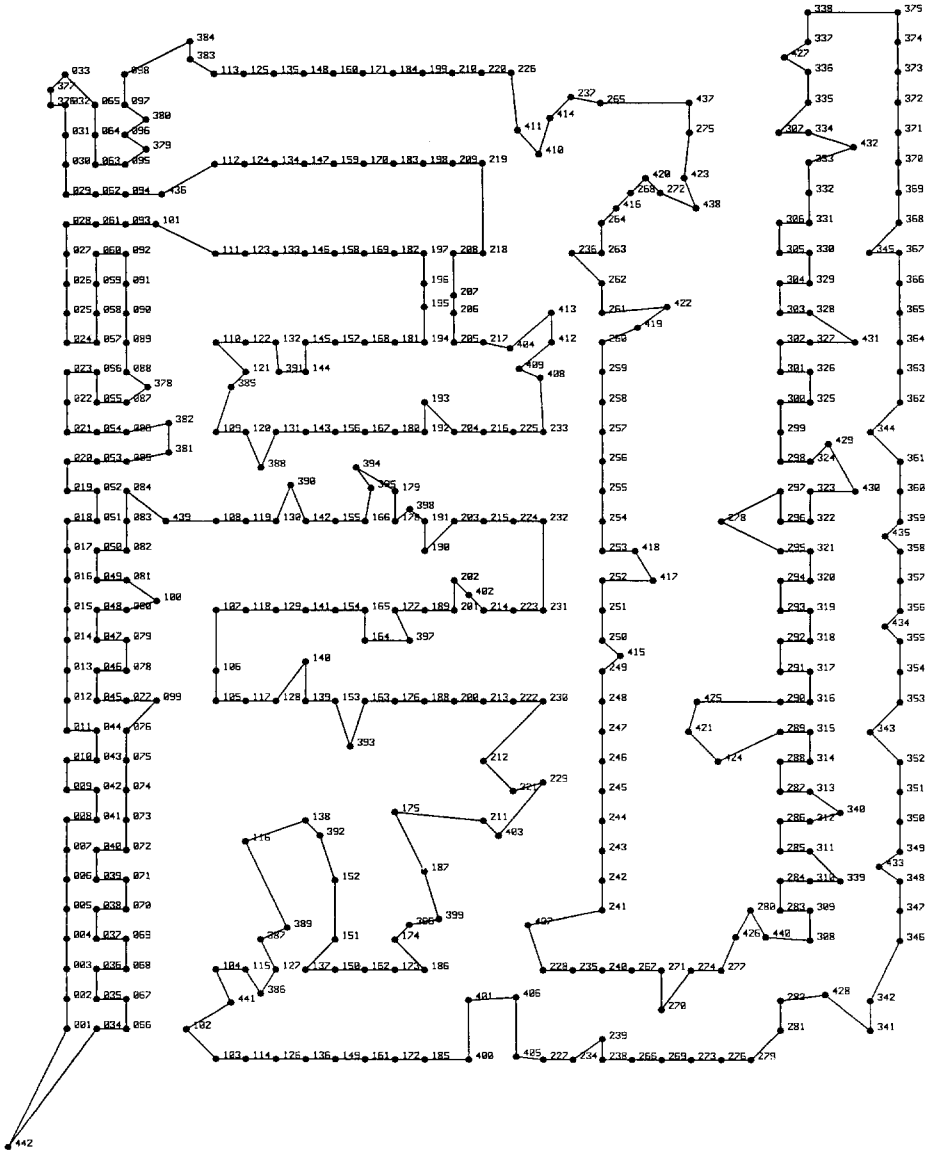
Picture 7.3. Optimal Europe-tour (202 cities).



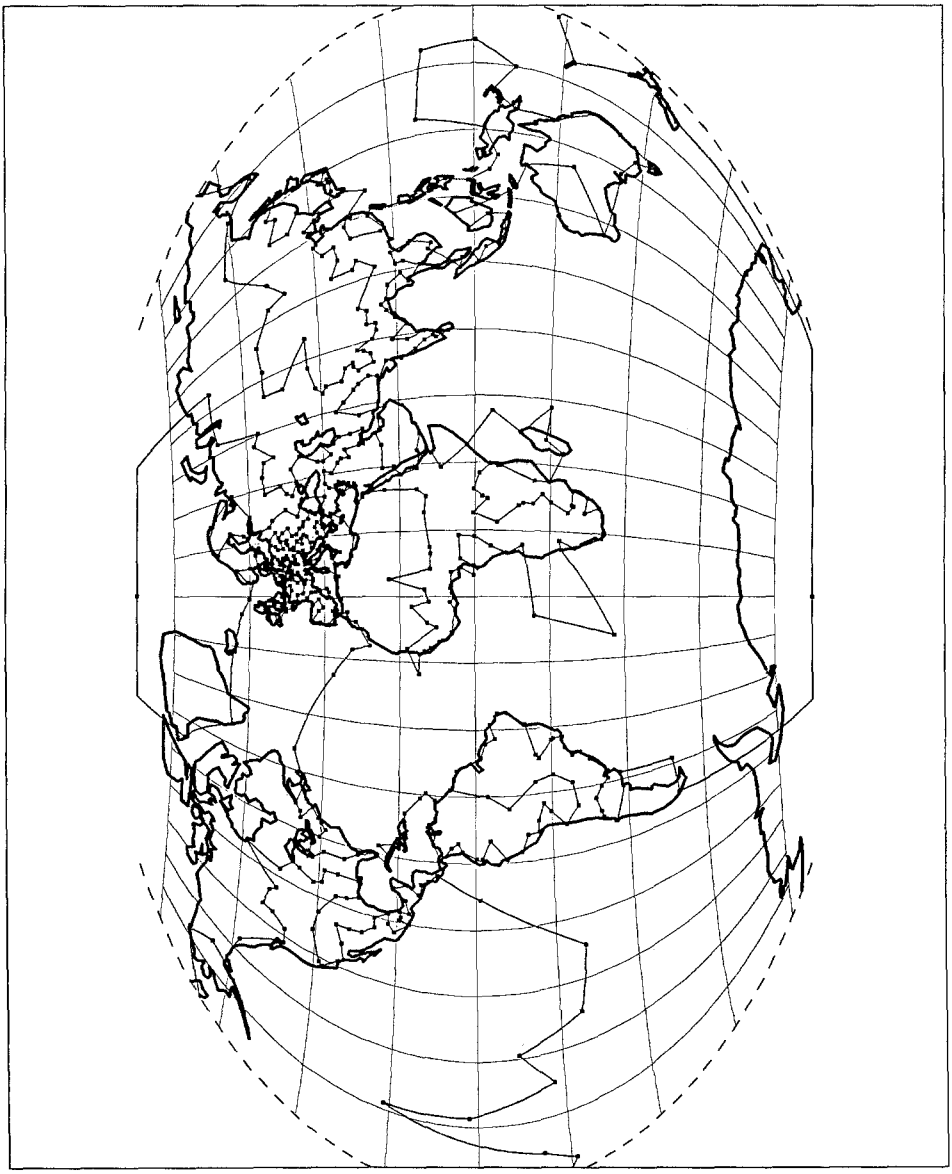
Picture 7.4. Optimal Australia–Asia-tour (229 cities).



Picture 7.5. Optimal Australia-Asia-Euruope-tour (431 cities).



Picture 7.6. Optimal tour of the 442-PCB-problem.



Picture 7.7. Optimal world-tour (666 cities).

Note added in proof

The data of the TSP examples mentioned in this paper are part of the library TSPLIB of traveling salesman problem instances. For a detailed description of this library cf.:

G. Reinelt: *TSPLIB - A Traveling Salesman Problem Library*, Report No. 250, Schwerpunktprogramm der Deutschen Forschungsgemeinschaft, Universität Augsburg, Augsburg, 1990, to appear in: *ORSA Journal on Computing.*

The library is available via E-Mail either from NETLIB or from the Computer and Information Technology Institute, Rice University.

(a) To get a description of the general NETLIB index use

mail netlib@ornl.gov
send index

(b) To get a description of available data at Computer and Information Technology Institute, Rice University, use

mail softlib@rice.edu
send README
send INDEX
send CATALOGUE

References

R.E. Bland and D.F. Shallcross, "Large travelling salesman problems arising from experiments in X-ray crystallography: a preliminary report on computation," Technical Report No. 730, School of OR/IE, Cornell University (Ithaca, NY, 1987).

H. Crowder and M.W. Padberg, "Solving large-scale symmetric travelling salesman problems to optimality," *Management Science* 26 (1980) 495-509.

G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, "Solution of a large scale traveling-salesman problem," *Operations Research* 2 (1954) 393-410.

J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices," *Journal of Research of the National Bureau of Standards B* 69 (1965) 125-130.

W. Felts, P. Krolak and G. Marble, "A man-machine approach toward solving the travelling-salesman-problem," *Communications of the ACM* 14 (1971) 327-334.

F. Glover, D. Klingman, J. Mote and D. Whitman, "A primal simplex variant for the maximum flow problem," Center of Cybernetic Studies, CCS 362 (Austin, TX, n.d.).

R.E. Gomory and T.C. Hu, "Multi-terminal network flows," *Journal of the Society for Industrial and Applied Mathematics* 9 (1961) 551-570.

M. Grötschel, *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme* (Hain, Meisenheim am Glan, 1977).

M. Grötschel and O. Holland, "Solving matching problems with linear programming," *Mathematical Programming* 33 (1985) 243-259.

M. Grötschel and O. Holland, "A cutting plane algorithm for minimum perfect 2-matchings," *Computing* 39 (1987) 327-344.

M. Grötschel, L. Lovász and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica* 1 (1981) 169-197.

M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization* (Springer, Berlin, 1988).

M. Grötschel and M. W. Padberg, "On the symmetric travelling salesman problem I: inequalities," *Mathematical Programming* 16 (1979) 265-280.

M. Grötschel and M.W. Padberg, "On the symmetric travelling salesman problem II: lifting theorems and facets," *Mathematical Programming* 16 (1979) 281-302.

M. Grötschel and M.W. Padberg, "Polyhedral theory," in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D. Shmoys, eds., *The Traveling Salesman Problem* (Wiley, Chichester, 1985) pp. 251-305.

M.W. Padberg and M. Grötschel, "Polyhedral computations," in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D. Shmoys, eds., *The Traveling Salesman Problem* (Wiley, Chichester, 1985) pp. 307-360.

M. Grötschel and W.R. Pulleyblank, "Clique tree inequalities and the symmetric travelling salesman problem," *Mathematics of Operations Research* 11 (1986) 537-569.

M. Held and R.M. Karp, "A dynamic programming approach to sequencing problems," *SIAM Journal on Applied Mathematics* 10 (1962) 196-210.

M. Held and R.M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Research* 18 (1970) 1138-1182.

M. Held and R.M. Karp, "The traveling-salesman problem and minimum spanning trees: part 2," *Mathematical Programming* 1 (1971) 6-25.

O. Holland, *Schnittebenenverfahren für Travelling-Salesman- und verwandte Probleme*, Doctoral Thesis, University of Bonn (Bonn, 1987).

R.L. Karg and G.L. Thompson, "A heuristic approach to solving travelling salesman problems," *Management Science* 10 (1964) 225-247.

S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research* 21 (1973) 498-516.

M.W. Padberg and M.R. Rao, "Odd minimum cut-sets and *b*-matchings," *Mathematics of Operations Research* 7 (1982) 67-80.

M.W. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric travelling salesman problem by branch and cut," *Operations Research Letters* 6 (1987) 1-7.

M.W. Padberg and G. Rinaldi, "An efficient algorithm for the minimum capacity cut problem," *Mathematical Programming* 47 (1990a) 19-36.

M.W. Padberg and G. Rinaldi, "Facet identification for the symmetric travelling salesman problem," *Mathematical Programming* 47 (1990b) 219-257.

T.H.C. Smith and G.L. Thompson, "A LIFO implicit enumeration search algorithm for the symmetric traveling salesman problem using Held and Karp's 1-tree relaxation," *Annals of Discrete Mathematics* 1 (1977) 479-493.