The Traveling Salesman Problem
Edited by E. L. Lawler, J. K. Lenstra,
A. H. G. Rinnooy Kan, D. B. Shmoys

9

Polyhedral computations

M. W. Padberg
New York University

M. Grötschel
Universität Augsburg

1 EQUIVALENCE OF OPTIMIZATION AND FACET IDENTIFICATION

This chapter focuses on the computational aspects of the polyhedral theory developed in the preceding chapter for the TSP. While in the previous chapter we were concerned with the problem of *describing* facet-inducing linear inequalities for the traveling salesman polytopes, we turn now to the problem of algorithmically *finding* facet-inducing linear inequalities. That is, we address the problem that one encounters if one wants to use the theoretical results of the preceding chapter in a cutting plane algorithm for the TSP using linear programming techniques.

After some introductory remarks concerning the number of facet-inducing linear inequalities and their algorithmic implications for the resolution of large-scale TSPs we discuss briefly the *equivalence* of optimization and

The first author's work was partially supported by the Università di Pisa and by the Consiglio Nazionale delle Ricerche, Italy.

(algorithmic) facet identification, a result which is based on the polynomial-time ellipsoid method of Khachian [1979] for linear programming problems, and which – in retrospect – has provided the theoretical backbone for the 'facet hunt' which began in the early 1970s for difficult combinatorial optimization problems [Padberg 1971, 1973; Pulleyblank 1973; Trotter 1973]. Then exact algorithms for the identification of subtour elimination and 2-matching constraints are given, as well as heuristic identification algorithms for comb inequalities. Finally, computational experiments using the above results are summarized. We use the same notation as in the previous chapter.

1.1 The number of facets of TSP polytopes and algorithmic implications

Except for the cases mentioned in Chapter 8, the exact numbers of facets of the traveling salesman polytopes are not known. One can, however, readily establish a (rather weak) upper bound on the number of facets for these polytopes. For instance, the polytope Q_T^n has at most

$$\binom{\frac{1}{2}(n-1)!}{\frac{1}{2}n(n-3)}$$

different facets. The bound is obtained by observing that there are $\frac{1}{2}(n-1)!$ different vertices (tours) of Q_T^n and that among those, possibly, any subset of $\frac{1}{2}n(n-3)$ vertices is linearly independent and thus generates a facet. However, the above bound is rather weak as some facets of Q_T^n contain considerably more than the minimum number of $\frac{1}{2}n(n-3)$ linearly independent tours needed in order to obtain a facet. In particular, one proves readily the following result.

Lemma 1 *Given an n-city symmetric TSP and a subtour elimination constraint $x(E(W)) \leqq |W|-1$, there are exactly $\frac{1}{2}w!(n-w)!$ tours satisfying $x(E(W)) = |W|-1$, where $w = |W|$, and $2 \leqq w \leqq n-2$.*

For the polytope P_T^n of the asymmetric TSP, one obtains similar statements quite easily.

Since the bound given above is rather weak, it is interesting to calculate the number of known different facets of the traveling salesman polytopes. We have done this for the polytope Q_T^n and the facets described in Theorem 14 of Chapter 8 for $n \leqq 120$. The respective numbers are given in Table 9.1.

For Q_T^{120} the number of nonequivalent subtour elimination and comb constraints has been computed exactly. Including the nonnegativity constraints, this number is exactly

26792549076063489375554618994821987399578869037768
70780484651943295772470308627340156321170880759399
86913459296483643418942533445648036828825541887362
427999209690792585547041772 87.

Table 9.1

No. of cities	No. of subtour elimination constraints cf. Theorem 14 (b) and (c) of Ch. 8	No. of comb constraints cf. Theorem 14 (d) of Ch. 8
6	25	60
7	56	2,100
8	119	42,840
9	246	667,800
10	501	8,843,940
15	16,368	1,993,711,339,620
20	0.5×10^{6}	1.5×10^{18}
30	0.5×10^{9}	1.5×10^{31}
40	0.5×10^{12}	1.5×10^{45}
50	0.5×10^{15}	10^{60}
120	0.6×10^{36}	2×10^{179}

The formulas to compute the respective numbers are given by Grötschel & Padberg [1979a]. We note that the numbers do not include the clique tree inequalities of Theorem 14(e) of Chapter 8, as at present we have no formula for the number of clique tree inequalities which are not comb inequalities. But, obviously, for reasonably large n there are many more possibilities to form clique tree inequalities than there are ways to form comb inequalities.

The simple story told by the above numbers is that, for the resolution of TSPs, linear programming techniques cannot be used in the *traditional* way which consists of listing explicitly all constraints over which we want to optimize. Ignoring for the moment the fact that we do not know a complete system of facet-inducing linear inequalities for the traveling salesman polytopes, it is already completely out of the question to *list* all *known* facet-inducing linear inequalities for the TSP and to solve the resulting linear program, because there are no computers available that are large enough to handle systems of linear inequalities of this size. In fact, there will never be such computers, since, for instance, the number of known facets of Q_T^{120} is about 10^{100} times the number of atoms of our universe. To abandon linear programming techniques for the resolution of TSPs on the basis of the above numbers – just because we cannot list and store all known facet-inducing linear inequalities – would be an equally erroneous conclusion to make, though a very common one, because historically, the exponential growth of the number of subtour elimination constraints is probably the reason why the seminal work by Dantzig, Fulkerson & Johnson [1954] has been ignored for such a long time. Indeed, in order to prove optimality of a vertex of Q_T^n, for example, all we really need is a *suitable* set of at most $\frac{1}{2}n(n-1)$ linear inequalities out of the immense universe of all possible facet-inducing linear inequalities.

Thus if we can prove optimality of a tour with respect to some set of at

most $\frac{1}{2}n(n-1)$ facet-inducing linear inequalities, by the validity of these inequalities for Q_T^n, optimality of this tour with respect to Q_T^n follows *a fortiori*. And what is most interesting, these considerations remain correct for any combinatorial optimization problem when we wish to optimize a given linear objective function.

The inevitable conclusion is thus that *suitable* constraints for the TSP (as well as for other combinatorial optimization problems with a linear objective function) must be generated 'on the fly', i.e. as computation progresses, if one wants to employ linear programming techniques in the resolution of large scale TSPs or any other hard combinatorial optimization problem. This idea for the TSP, which was implemented by Dantzig, Fulkerson & Johnson [1954], gave rise to the cutting plane approach to integer programming and has led to the famous algorithmic proof of finite solvability of integer programming problems by Gomory [1963]. However, the cutting planes considered by Gomory – while guaranteeing finite convergence of the procedure – in general do not belong to a system of facet-inducing linear inequalities. As a result, convergence tends to be slow and, more importantly from a computational point of view, because of the great number of the nonzero coefficients of the cutting planes which have to be stored during the calculation, storage requirements simply explode.

It therefore appeared necessary to consider mathematically proven *good* cutting planes, namely facet-inducing linear inequalities, if the cutting plane idea was useful at all in actual computation. Every facet-inducing linear inequality belongs to a minimal system of linear inequalities describing the convex hull of the solution set of an integer program (cf. Section 1 of Chapter 8). Facet-inducing inequalities are the strongest possible valid cutting planes and hence the most useful cutting planes in linear programming based calculations, while arbitrary valid inequalities need not necessarily be considered in the solution of integer programming problems and thus should be avoided.

The problem we want to solve in an iterative procedure for the resolution of TSPs (or, likewise, for any other combinatorial optimization problem with linear objective function) using linear programming techniques, can be formulated in general terms as follows. As in Section 2.1 of Chapter 8, let E be a finite ground set and $\mathcal{F}$ be a nonempty set of subsets of E, and let

$$P_{\mathcal{F}} := \operatorname{conv}\{x^F \in \mathbb{R}^E \mid F \in \mathcal{F}\}$$

be the polytope given by the convex hull of the incidence vectors x^F of the elements of $\mathcal{F}$.

Facet identification problem *Given a point $\bar{x} \in \mathbb{R}^E$ and a polytope $P_{\mathcal{F}}$, find a facet-inducing linear inequality $f^T x \leqslant f_0$ for $P_{\mathcal{F}}$ which is violated by $\bar{x}$, i.e. such that $f^T \bar{x} > f_0$, or prove that no such inequality exists, i.e. that $\bar{x} \in P_{\mathcal{F}}$.*

We can now state a general procedure for the resolution of combinatorial optimization problems using linear programming calculations which we

could call a (dual) cutting plane procedure, but which we prefer to call a *relaxation* procedure because what one solves is indeed a sequence of relaxations of the underlying combinatorial problem.

Relaxation method for combinatorial problems

Step 1. (*Initialization*) Let (LP_0) be a valid linear programming relaxation of the combinatorial problem under consideration; set $k = 0$.
Step 2. (*LP solver*) Solve (LP_k); let x^k be an optimal solution to (LP_k).
Step 3. (*Facet identification*). Solve the facet identification problem for x^k and the polytope $P_{\mathcal{F}}$.
  Step 3.1. If one or more violated facet-inducing linear inequalities for $P_{\mathcal{F}}$ are found, define (LP_{k+1}) to be (LP_k) amended by the violated facet-inducing inequalities, set $k := k+1$ and go to Step 2.
  Step 3.2. If a violated facet-inducing linear inequality does not exist, stop.

If a *finite* method for solving the linear programs is used in Step 2, the finiteness of the above method follows from the finiteness of the number of facet-inducing linear inequalities which 'in principle' can simply be listed and checked off one by one.

In an iterative procedure for the symmetric TSP, for instance, we can thus start with a linear programming relaxation of the problem where the objective function is specified by the given distance function c and the constraint set is initially empty or is the one given by the constraints

$$\begin{aligned} Ax &= 2, \\ 0 \leqslant x_e &\leqslant 1 \qquad \text{for all } e \in E, \end{aligned} \tag{1}$$

where A is the vertex–edge incidence matrix of the complete undirected graph having n vertices. Now one proves readily the following result.

Lemma 2 *If the simplex method is used in Step 2 of the relaxation method, then an incidence vector $\bar{x}$ corresponding to an optimal tour is obtained in a finite number of iterations provided that a complete system of facet-inducing linear inequalities for Q_T^n is known.*

Of course, the obvious trouble with the relaxation method is that we do not know all the facet-inducing inequalities of Q_T^n or, more generally, of an arbitrary polytope $P_{\mathcal{F}}$. As a consequence we have to resort to branch and bound (or to some other enumerative technique) at some point during the calculation.

Surprising as it may seem, this idea works well in computational practice. The pertinent computational results for the TSP are summarized in Section 4 of this chapter. It should also be noted that the above approach has yielded good computational results for the optimization of sparse large-scale 0–1 linear programming problems having no special structure [Crowder, Johnson & Padberg, 1983], as well as for some other combinator-

ial optimization problems [Barahona & Maccioni, 1982; Grötschel, Jünger & Reinelt, 1984].

A less obvious drawback of the relaxation method is the following one. Even if a complete list of facet-inducing inequalities is known (for the TSP or any other combinatorial optimization problem with linear objective function), it is not obvious that the facet identification problem can be solved efficiently. Of course, it can evidently be solved *enumeratively* by listing all facet-inducing inequalities and checking them one by one for violation by the given point $\bar{x}$. But in most cases, even for polynomially solvable problems, the number of facets is exponential in the problem data (see Section 3 of Chapter 8); thus – disregarding possible difficulties with Step 2 – the linear programming based algorithm would be bad because Step 3 requires exponential time.

Thus, the question is: Under what conditions can the facet identification problem be solved by a good algorithm? The next two sections of this chapter give a satisfactory general answer to this fundamental question. We note for completeness that the facet identification problem has been solved by means of good algorithms for the b-matching problem with and without upper bounds by Padberg & Rao [1982], for the spanning tree problem by Padberg & Wolsey [1983], and for the matroid optimization problem by Cunningham [1984].

1.2 The ellipsoid method for linear programs

To discuss the fundamental relationship between optimization and facet identification we first review briefly the ellipsoid method for linear programs as developed by Khachian [1979]. (For a recent survey on computational aspects of the method we refer the reader to Bland, Goldfarb & Todd [1981] or Schrader [1982].) In the next section we discuss some of the theoretical consequences that this polynomially bounded algorithm for linear programming problems has for combinatorial optimization problems like the TSP. The material of this section and the next one is based on papers by Grötschel, Lovász & Schrijver [1981], Karp & Papadimitriou [1982] and Padberg & Rao [1985], though we will follow most closely the last paper.

To state the ellipsoid method for linear programs, we consider the linear program

$$\max\left\{\sum_{j=1}^{n} c_j x_j \;\middle|\; Ax \leqslant b\right\} \tag{2}$$

and denote the ith row of (A, b) as (a^i, b_i) for $i = 1, \ldots, m$. We assume that all data are *integers*, i.e. rational data are converted to integers by appropriate scaling, and, for simplicity, that A has full column rank which is the case, for example, if the nonnegativity constraints $x_j \geqslant 0$ for $j = 1, \ldots, n$ are among the constraints of (2). Let P denote the feasible set of (2) and

suppose that we want to maximize $c^T x$ over the set

$$S = P \cap \left\{x \in \mathbb{R}^n \;\middle|\; \sum_{j=1}^{n} x_j^2 \leqslant u^2\right\},$$

where u is some positive number chosen to be large enough such that S contains all *basic* solutions to $Ax \leqslant b$. We define

$$\begin{aligned} z_L &= -1 + u \sum_{j=1}^{n} \min\{0, c_j\}, \\ \bar{c} &= 1 + \max\{|c_j|, j = 1, \ldots, n\}, \\ h &\geqslant 2\bar{c}n^2\Delta_A, \qquad h \text{ integer}, \\ t &= 4n^2\lceil \log_2 h^2 u n^{1/2}\rceil, \\ R &= 16n\lceil \log_2 h u^{1/2}\rceil, \end{aligned}$$

where $\Delta_A \geqslant 2$ is an integral upper bound on the absolute values of the determinants of the submatrices of the matrix A.

The following algorithm, ALG1(u, h), is one version of the ellipsoid method with approximate arithmetic for linear programs with integer data.

The ellipsoid method for linear programs ALG1(u, h)

Step 1. (*Initialization*) Set $x_j^0 = 0$ for $j = 1, \ldots, n$; $Q_0 = uI_n$ where I_n is the $n \times n$ identity matrix; $\bar{z} = z_L$, $k = 0$.

Step 2. (*Constraint identification*) If $a^i x^k \leqslant b_i + 1/h$ for all $i = 1, \ldots, m$, go to Step 3. Else let j be any index with $a^j x^k > b_j + 1/h$. Set $r = a^j$ and go to Step 4.

Step 3. (*Objective function as a constraint*) If $cx^k > \bar{z}$, replace $\bar{z}$ by cx^k and set $\bar{x} = x^k$. Set $r = -c$.

Step 4. (*Updating*) If $k = t$, go to Step 5. Else set

$$x^{k+1} = x^k - Q_k Q_k^T r^T/((n+1)\,\|rQ_k\|),$$

$$Q_{k+1} = 2^{1/Rn^2}(n/\sqrt{n^2-1})Q_k\left(I_n - \left(1 - \sqrt{\frac{n-1}{n+1}}\right)Q_k^T r^T rQ_k/\|rQ_k\|\right),$$

where $=$ means that for each component the computation is carried out with a precision of R binary positions after the point. Set $k = k + 1$ and go to Step 2.

Step 5. (*Termination*) If $\bar{z} = z_L$, stop; the problem has no feasible solution. Else stop; $\bar{x}$ approximately solves the linear programming problem.

The algorithm ALG1(u, h) thus executes t iterations and, apart from the space requirements for storing the constraints of (2), it requires $O(n^2R)$ bits of workspace. Both the running time and the workspace requirements are bounded by a polynomial function of n, $\log_2 h$ and $\log_2 u$. The parameters u

and h can assume different values, and the implications of different parameter settings are discussed next.

Let $D \geq 2$ be an integral upper bound on the absolute value of a determinant of a submatrix of (A, b). Then, if $\bar{x}$ is a basic solution to $Ax \leq b$, we have $|\bar{x}_j| \leq D$ for all $j = 1, \ldots, n$, and furthermore, $D \geq \Delta_A$ holds. Running the algorithm ALG1(u, h) with the parameters

$$u = 2n^{3/2}\bar{c}D^3, \qquad h = 2n^2\bar{c}\Delta_A^2,$$

one can prove that (2) has an unbounded optimum solution if and only if the algorithm terminates with an objective function value $\bar{z} \geq n\bar{c}D$. This follows because we assume integrality of the data and because the initial ellipsoid contains all points $x \in \mathbb{R}^n$ satisfying

$$-2n\bar{c}D^3 \leq x_j \leq 2n\bar{c}D^3 \qquad \text{for } j = 1, \ldots, n.$$

If we conclude infeasibility or unboundedness, we have solved (2). Else, we run ALG1(u, h) a second time with the additional upper and lower bounds $|x_j| \leq D$ for $j = 1, \ldots, n$, and the following parameters:

$$u = \sqrt{n}(D+1), \qquad h = 2n^2\bar{c}\Delta_A^3.$$

Assuming for simplicity that (2) has a unique optimum over the feasible set of (2) – alternatively, we perturb the objective function of (2) slightly, as in Exercise 6 below – one can prove that we get an optimal solution to (2) which is basic to the system of linear inequalities $Ax \leq b$, $-D \leq x_j \leq D$ for $j = 1, \ldots, n$ by rounding the final trial solution $\bar{x}$ obtained from ALG1(u, h) using the method of continued fractions. If the optimal solution x obtained this way satisfies $-D < x_j < D$ for all $j = 1, \ldots, n$, then we are done. Else, because of the perturbation device mentioned above, the optimum of the perturbed objective function may have become unbounded over the constraint set $Ax \leq b$ of (2) even though the original objective function is bounded, i.e. some of the components of the solution x may be equal to $\pm D$. However, one can show that starting from the solution x obtained by ALG1(u, h), one can find an optimal basic solution to (2) with a computational effort which remains polynomially bounded in the desired parameters. Thus by running the ellipsoid method ALG1(u, h) at most twice we can solve the linear program (2).

Theorem 1 *There exists an algorithm based on the ellipsoid method* ALG1(u, h) *which determines infeasibility or unboundedness of* (2) *or finds an optimal basic solution to* (2) *and whose time and workspace requirements depend linearly on the number m of constraints of* (2) *and polynomially upon the problem parameters n,* $\log_2 \bar{c}$ *and* $\log_2 D$.

The length L of the input data of (2) for a binary computer is at least

$$L = 1 + \log_2(mn) + \sum_{i=1}^{m} \log_2(|b_i|+1) + \sum_{j=1}^{n} \log_2(|c_j|+1) + \sum_{i,j} \log_2(|a_{ij}|+1).$$

Using Hadamard's inequality (cf. Marcus & Minc [1964]) to estimate D, it can be shown that L is bounded from below by a polynomial function of the parameters m, n, $\log_2 \bar{c}$ and $\log_2 D$ of Theorem 1. Thus the theorem implies the existence of a polynomial algorithm for (2) in the usual sense of the definition of polynomiality of algorithms [Garey & Johnson, 1979].

1.3 Facet identification and optimization

The fact that the ellipsoid method for linear programs does *not* depend on the input length L of (2), but rather upon the parameters specified in Theorem 1, makes it a possible instrument for answering the fundamental question raised in Section 1.1 concerning the relationship between optimization and facet identification. As pointed out earlier, the linear programming formulation of most combinatorial problems involves a number m of constraints which depends exponentially upon the number n of variables, thus producing linear programs of the form (2) whose length L grows exponentially with n, as in the case of the TSP. The parameters other than m appearing in Theorem 1, however, can be estimated using n and the largest absolute value of the numbers of the data of (2), and are thus independent of the actual number of constraints. In order to (theoretically) successfully apply the ellipsoid method to combinatorial optimization problems, we must thus be able to

(i) remove the explicit dependence on the number m of constraints of the linear programming formulation of the combinatorial problem; and
(ii) prove that the coefficients of the linear programming formulation of the combinatorial problem remain reasonably small.

If both difficulties can be removed, we can prove the existence of polynomial algorithms for combinatorial optimization problems, under the assumption that complete inequality systems for the associated polytopes are known. As we shall see, the second problem (ii) is a minor one, while the first one is indeed equivalent to the solvability of a combinatorial optimization problem by a good algorithm.

Consider now the combinatorial optimization problem

$$\max\{p^T x \mid x \in P_S\} \tag{3}$$

where $p \in \mathbb{R}^E$ is a vector with integer components p_j for $j = 1, \ldots, |E|$ and P_S is defined in Section 1.1. For notational simplicity denote $n = |E|$. Let $r \leq n$ denote the dimension of P_S. In this section only, we shall call any valid inequality which is satisfied by *at least* r affinely independent 0–1 points of P_S a *facet* of P_S. This definition differs slightly from our earlier one where *at least* is replaced by *exactly*. It is more convenient to work here with this definition because it means that we need not distinguish between the equations and inequalities which define P_S (cf. Section 1 of Chapter 8).

We will assume without loss of generality that P_S is nonempty. If we let

$z_U \geq 0$ (z_L, respectively) denote a proper upper (lower, respectively) bound on the linear form $p^T x$ over $0 \leq x_e \leq 1$ for all $e \in E$, we can introduce a new 0–1 variable x_{n+1} having an objective function coefficient $p_{n+1} = z_L - z_U$. (Remember: $n = |E|$.) If we modify E by adjoining a new element $n+1$ and changing $\mathcal{F}$ by declaring all incidence vectors $(x^F, 0)$ with $F \in \mathcal{F}$ and the vector $(\mathbf{0}, 1)$ *feasible* where $\mathbf{0}$ is the vector with n components equal to 0, the problem in $n+1$ variables always has a feasible solution. The original problem is feasible if and only if the objective function value over the problem in $n+1$ variables is greater than z_L. Increasing the dimension of the problem by one does not change the question we are addressing in any significant way.

As we know (for instance, from Weyl [1935]), we can describe $P_{\mathcal{F}}$ by a system of linear inequalities. More precisely, one can prove the following theorem.

Theorem 2 *There exists a system of facet-defining linear inequalities for $P_{\mathcal{F}}$ such that the absolute value of each component is an integer less than or equal to*

$$\Xi = n^{n/2}.$$

It follows from Theorem 2 that for a suitably chosen system of facet-defining inequalities for $P_{\mathcal{F}}$, the logarithm (base 2) of the absolute value of a nonzero coefficient is bounded by a polynomial in the number of variables n and this answers problem (ii) above completely.

We will thus use the term *facet-defining inequality for $P_{\mathcal{F}}$* to mean only facet-defining inequalities with integer coefficients whose absolute values are less than or equal to Ξ.

Denote by $Ax \leq b$ a complete system of facet-defining inequalities for the polytope $P_{\mathcal{F}}$. Then problem (3) is equivalent to the linear program

$$\max\left\{\sum_{j=1}^{n} p_j x_j \;\middle|\; Ax \leq b\right\}.$$

In order to get a unique optimum solution to this linear program, perturb the objective functions coefficients as follows:

$$c_j = 2^n p_j + 2^{n-j}, \qquad \text{for } j = 1, \ldots, n,$$

and consider the corresponding linear program with integer data,

$$\max\left\{\sum_{j=1}^{n} c_j x_j \;\middle|\; Ax \leq b\right\}. \tag{4}$$

Let $\Delta_A \geq 2$ be an integral upper bound on the absolute value of a determinant of a submatrix of A. With the above convention it follows from Hadamard's inequality that

$$\Delta_A \leq n^{n/2}\Xi^n = n^{(n^2+n)/2}$$

holds. Let z_L be a proper lower bound on the optimum value of (4) and let $\bar{p} = 1 + \max\{|p_j|,\ j = 1, \ldots, n\}$.

It then follows by choosing the parameters

$$u = 2\sqrt{n}, \qquad h = \bar{p} n^2 2^{n+2} \Delta_A$$

that the ellipsoid method ALG1(u, h) finds an approximate optimal solution $\bar{x}$ to (4) in time and workspace requirements which depend polynomially on n, $\log_2 \bar{p}$ and linearly on m, the number of constraints of (4). Moreover, the approximate solution $\bar{x}$ to (4) can be rounded to an optimal integer solution x^* to (3) by setting $x_j^* = \lfloor \bar{x}_j + 0.5 \rfloor$ for $j = 1, \ldots, n$.

As discussed previously, the dependence on the number m of constraints of (4) is linear. It is clear that the entire computational effort for the solution of (4) is bounded by a polynomial function of n and $\log_2 \bar{p}$ if the constraint identification in Step 2 of the ellipsoid algorithm can be carried out by some *subroutine* whose computational effort is bounded by a polynomial function of n and $\log_2 \bar{p}$. The constraint identification step, however, is exactly the facet identification problem – except that we permit a small error given by the term $1/h$ in the respective constraints. The vector $\bar{x}$ for which the facet identification has to be carried out is a *rational* vector whose components require $O(R)$ bits in the binary representation, where R is the required precision of the algorithm and R is a polynomial function in the above parameters. Thus if the facet identification problem can be solved for a rational vector $\bar{x}$ and the polytope $P_{\mathcal{F}}$ by an algorithm whose running time and workspace requirements depend polynomially upon n, $\log_2 \bar{p}$ and the length of the binary encoding of $\bar{x}$, then it follows that (3) can be optimized by an algorithm which is polynomial in the desired parameters, i.e. we can resolve problem (i) introduced above.

The preceding discussion partially settles the question concerning the relationship between facet identification and optimization. It is a most interesting fact – whose proof is too technical to be discussed here – that the above statement can be reversed as well.

Theorem 3 *The combinatorial optimization problem* (3) *can be solved by a polynomial algorithm for any integer vector p if and only if the facet identification problem can be solved by a polynomial algorithm for any rational vector $\bar{x}$.*

Thus Theorem 3 can be used to prove the existence of good algorithms for (3) by exhibiting good algorithms for the facet identification problem, and vice versa. For instance, the above theorem implies the existence of a good algorithm for the facet identification problem of the vertex packing problem in claw-free graphs because there is a polynomial algorithm for such problems [Minty, 1980]. Yet preliminary studies of the associated polytope by Giles & Trotter [1981] have shown how complex this polytope is, while leaving the facet identification problem unanswered. Its solution remains an

interesting challenge. On the other hand, the facet identification problem has been resolved satisfactorily for the b-matching problem, the spanning tree problem and the matroid optimization problem, thus yielding polynomial algorithms for these problems which are different from the previously known ones. As we shall see, this has interesting implications for the TSP.

While our discussion here has concentrated on finding facets of the underlying polytope $P_\#$, it is clear from the discussion that we need not restrict ourselves to facet-defining inequalities. Two properties, however, are crucial to the argument, namely: (i) the totality of the linear constraints considered must define the polytope $P_\#$ (or do so at least locally for some optimizing point of $P_\#$), and (ii) the length of the coefficients of the linear constraints *must not* grow exponentially in the problem parameters. Thus any set of *valid* linear inequalities satisfying (i) and (ii) permit us to arrive at the same conclusion. Using this fact, Grötschel, Lovász & Schrijver [1981] have derived polynomial algorithms for various problems on perfect graphs. It is therefore worthwhile to keep in mind that *for theoretical purposes* it suffices to work with linear inequalities that do the job. However, as the experience with earlier cutting plane methods has demonstrated, practical computational considerations suggest that best possible linear inequalities, facet-defining linear inequalities, should be used because the latter are the *tightest* constraints possible and typically have the additional advantage of sparsity of the nonzero coefficients, thus permitting storage requirements to be kept at a minimum.

1.4 General computational considerations

The polynomial-time ellipsoid method for linear programs has permitted us to answer the fundamental question concerning the relationship between optimization and facet identification, thus putting linear programming techniques for the solution of combinatorial optimization problems on a sound theoretical basis.

However, given the current state of the development of the ellipsoid method, it is an undisputed fact that the simplex method is superior in numerical computation. Part of the reason for this statement rests with the fact that the polynomial bound applies to the worst-case situation. Practical problems arising in combinatorial problem solving hardly qualify for the attribute 'worst-case'. Furthermore, the analysis of algorithms is an asymptotic theory, and therefore, is concerned with the behavior of algorithms for large problem sizes. More importantly, sparsity of the constraint matrix, special structure of the constraint matrix, the possibility of using heuristically obtained solutions to speed up convergence, and so forth, are approaches which by now, 35 years after the invention of the simplex method, are integrated to a very large degree into existing software for the solution of linear programming problems. Mathematical programming software products, such as CDC's code APEX, IBM's software system MPSX-MIP/370, Ketron's MPSIII, SCICON's code SCICONIX, Sperry Univac's FMPS, etc., are all highly developed products, and are the outcome of years of effort that combined the skills of many mathematicians, computer scientists and computer programmers. It is therefore not only natural but also advisable to use such software systems as building blocks for the practical solution of difficult combinatorial optimization problems. More precisely, in Step 2 of the relaxation method for combinatorial optimization problems we propose to use the *most efficient* linear programming solver that is available.

This implies in particular that we favor a *dual* cutting plane approach over the *primal* cutting plane approach, though this is partly due to the fact that a primal cutting plane approach requires a different set of linear programming tools than are available. (For the terminology we refer the reader to Garfinkel & Nemhauser [1972].) The intuitive appeal of a primal cutting-plane procedure stems from the fact that for many practical combinatorial optimization problems, heuristics can be used to find a good or sometimes even, unknowingly, an optimal solution to the problem at hand. In an attempt to *prove optimality* of the heuristically obtained solution x, one can therefore start the linear programming calculations at the extreme point x of $P_\#$ obtained heuristically, and iterate by cutting off adjacent fractional solutions by hyperplanes which contain x, until either optimality of x is proven or a better adjacent integer solution is found. In the latter case, one restarts the whole procedure at the new integer solution. If a complete list of facet-inducing linear inequalities is known, the procedure can be implemented, for example, in the framework of a modified simplex method, and one need only solve the following facet identification problem.

Primal facet identification problem *Given a point $\hat{x}\in\mathbb{R}^N$ and an extreme point $x^*\in\mathbb{R}^N$ of the polytope $P_\#$, find a facet-inducing linear inequality $f^Tx\leqslant f_0$ for $P_\#$ such that $f^T\hat{x}>f_0$ and $f^Tx^*=f_0$ hold, or prove that no such inequality exists.*

We note that for the cases where the facet identification problem has been solved by exact algorithms, the same methods apply to answer the primal facet identification problem (see Exercise 7).

In a computational study [Padberg & Hong, 1980] a primal cutting plane approach was employed and this required the writing of a simplex code of their own. In more computational studies [Grötschel, 1977a, 1980b; Crowder & Padberg, 1980; Crowder, Johnson & Padberg, 1983] a dual cutting plane approach was adopted and IBM's system MPSX/370 was used to solve the linear programs. Based on this experience we prefer the dual cutting plane approach, last but not least since the dual approach permits one to treat the linear programming routine as a 'black box' in combinatorial problem solving.

This leaves the question of how to solve the facet identification problem. In order to make the solution of this problem operational we distinguish

different types or classes of facet-inducing linear inequalities. The subtour elimination constraints form a first class of facets of the TSP, the 2-matching constraints a second class, the comb constraints a third class, and so forth. The reason that we make such a distinction is not because of the chronological order in which these classes of facet-inducing inequalities were found, but rather, as we shall see, *different algorithms* are needed to solve the facet identification problem for these different classes of facet-inducing inequalities. This means that Step 3 of the relaxation method for combinatorial optimization problems must be thought of as a whole set of different subroutines which are called upon when needed. Thus we could dub the whole preceding a *multi-algorithmic approach* to combinatorial problem solving.

It is clear that the basic scheme of the procedure is rather simple to implement. For, regardless of the *number* of subroutines required in Step 3 of the relaxation method, once the data structures are set up for a single subroutine to do the facet identification, we simply add more and more subroutines as our knowledge of exact (or heuristic) solution procedures for the identification of different types of facets increases. Furthermore, since the basic outline is the same for any combinatorial optimization problem, we can use the same main routine for quite different combinatorial optimization problems simply by exchanging the subroutines used for the facet identification phase.

Since currently available linear programming software permits the user to revise the linear program by generating, for instance, new constraints, the implementation of the above ideas is feasible with almost any software system. However, it appears that there are inefficiencies in the present-day codes concerning the addition of new constraints. These inefficiencies do not prohibit the implementation, but it appears that software systems especially designed to handle constraint generation would enhance the performance. Most linear programming software systems were designed originally with a fixed number of rows in mind, where the addition of new rows was the exception rather than the rule, as is the case here.

Exercises

1. Prove Lemma 1.
2. Given any linear program (LP) with p nonnegative variables and q linear constraints having a finite optimum solution, prove that there exists a linear program (LP*) with (at most) p linear constraints obtained from (LP) by dropping (at least) q of the $p+q$ linear constraints of (LP) such that every optimal solution to (LP) is also an optimal solution to (LP*). Moreover, every optimal basic solution to (LP*) is also an optimal basic solution to (LP).
3. Prove Lemma 2.
4. Prove the more general statement analogous to Lemma 2 for an arbitrary

polytope $P_{\mathcal{I}}$ associated with any combinatorial optimization problem having a linear objective function (cf. Section 2.1 of Chapter 8).
5. (a) Prove that there exists a system of facet-defining inequalities for the traveling salesman polytope Q_T^n such that each coefficient is a nonnegative number less than or equal to $n^{n(n-3)/4}$. (b) Prove Theorem 2 if $\dim(P_{\mathcal{I}}) = n$ and $\mathcal{I}$ is an independence system. (*Hint:* Give an extreme point characterization of the facet-defining inequalities of Q_T^n ($P_{\mathcal{I}}$, respectively). Then use Cramer's rule and Hadamard's inequality.)
6. Prove that (4) has a unique optimum solution and that the optimum solution to (4) is optimum for (3).
7. (a) Show that the primal facet-identification problem can be reduced to solving the facet-identification problem for some point $\bar{x} = (1-\mu)x^* + \mu\hat{x}$ where $0 < \mu \leq 1$. (b) Using the notation of Section 1.3 and assuming that $\hat{x} \in \mathbb{R}^E$ has rational components, give a formula for a value μ of part (a) that works.

2 IDENTIFICATION OF SUBTOUR ELIMINATION CONSTRAINTS

When restricted to subtour elimination constraints, the facet-identification problem can be formulated in the case of the TSP as follows, where $G = (V, E)$ is the complete graph on n vertices.

Subtour problem *Given a point $\bar{x} \in \mathbb{R}^E$ satisfying $0 \leq \bar{x}_e \leq 1$ for all $e \in E$, find a set $W \subseteq V$, $2 \leq |W| \leq n-1$, such that $\bar{x}(E(W)) > |W| - 1$ holds, or prove that no such $W \subseteq V$ exists.*

Note that we have assumed for simplicity that the point $\bar{x} \in \mathbb{R}^E$ satisfies $0 \leq \bar{x}_e \leq 1$ for all $e \in E$ because there are only $2|E|$ such constraints and thus they can be checked in time polynomial in $|E|$.

In the asymmetric case we have the same problem except that E is replaced by A throughout in the subtour problem. Using the mapping $f(y) = x$ defined in Section 5.2 of Chapter 8, one reduces the problem of finding a violated subtour elimination constraint for the asymmetric TSP to the subtour problem by replacing each pair of arcs (i, j) and (j, i) by the (undirected) edge $e = \{i, j\}$ and giving this edge the weight $\bar{x}_e = \bar{x}_{ij} + \bar{x}_{ji}$.

Lemma 3 *Let $\bar{x} \in \mathbb{R}^A$ be given and let $\bar{\bar{x}} \in \mathbb{R}^E$ be obtained by the above symmetrization.*
(a) *There exists a subtour elimination constraint for P_T^n which is violated by $\bar{x}$ if and only if there exists a subtour elimination constraint for Q_T^n which is violated by $\bar{\bar{x}}$.*
(b) *If $\bar{x} \in \mathbb{R}^A$ satisfies $\bar{x}(\delta(v)) = \bar{x}(\delta(v)) = 1$ for all $v \in V$, then $\bar{\bar{x}}(\delta(v)) = 2$ holds for all $v \in V$.*
(c) *Let $\bar{x} \in \mathbb{R}^E$ satisfy (1) and let $x^* \in \mathbb{R}^E$ be the incidence vector of a tour, i.e. x^* is an extreme point of Q_T^n. Then there exists a set $W \subseteq V$ with $2 \leq |W| \leq n-1$ such that $\bar{x}(E(W)) > |W| - 1$ and $x^*(E(W)) = |W| - 1$ hold if and only if*

there exists $W \subseteq V$ *with* $2 \leqslant |W| \leqslant n-1$ *such that* $\bar{x}(E(W)) > |W| - 1$ *holds, where* $\bar{x} = \frac{1}{2}\mathbb{1} + \frac{1}{2}x^*$.

Thus we need to consider only the case of the symmetric TSP.

As part (c) of Lemma 3 shows, the primal facet identification problem when restricted to subtour elimination constraints, can be reduced to the subtour problem as well. Exercise 9 extends the above symmetrization technique to constraints other than subtour elimination constraints.

The facet identification problem for the 1-tree polytope Q^n_{1T} and its monotonization $\tilde{Q}^n_{1T}$ (cf. Section 3.1 of Chapter 8) is closely related to the foregoing and can be broken into two parts. First, there are only polynomially many constraints involving the special vertex 1 of the underlying graph. Thus they can be checked in polynomial time. Second, the exponentially many constraints (14) of Section 3.1 of Chapter 8 do not involve the special vertex 1. We can thus delete vertex 1 and all edges incident to it. The facet identification problem for Q^n_{1T} and $\tilde{Q}^n_{1T}$ respectively, then becomes precisely the subtour problem on the smaller graph *except* that W is permitted to equal the full vertex set of the smaller graph. In terms of the subtour problem, this means that the cardinality restriction $|W| \leqslant n-1$ must be dropped to deal with this case. Furthermore, in the case of the branching polytope $\tilde{P}^n_B$, the antibranching polytope $\tilde{P}^n_{\bar{B}}$, the arborescence polytope P^n_B and the antiarborescence polytope $P^n_{\bar{B}}$ (cf. Section 3.2 of Chapter 8), one proceeds likewise by dropping the two special vertices 1 and $n+1$ and using the symmetrization technique described above. In this way, the facet identification problem for the exponentially many constraints (28) of Chapter 8 becomes precisely the problem we have just discussed.

We discuss in Section 2.1 a *heuristic* procedure for the solution of the subtour problem and its variant as necessary for the 1-tree polytope. In many cases this heuristic solves the problem. Moreover, if it does not solve it, then it frequently brings about a reduction in the *size* of the problem which has to be solved by an exact algorithm. In Section 2.2 we state *exact* algorithms for the identification of subtour elimination constraints: The first one deals with the specific case of the symmetric TSP; that is, we assume that the point $\bar{x} \in \mathbb{R}^E$ of the subtour problem satisfies the equations of (1). In particular, here we are looking for a *nonempty proper* subset $W \subset V$ which does the job, or we want to prove that no such subset exists. The second algorithm addresses the problem for the 1-tree polytope where the point $\bar{x} \in \mathbb{R}^E$ need not satisfy the equations of (1) and where we are looking for a *nonempty* set $W \subseteq V$ which does the job. In this case, however, W may be equal to V.

2.1 A heuristic for subtour elimination constraints

Let $\bar{x} \in \mathbb{R}^E$ satisfy $\bar{x}_e \geqslant 0$ for all $e \in E$. Interpret $\bar{x}_e$ as the *weight* of edge e. Edges having a weight of 0 are of no interest to us here and we denote by $G(\bar{x}) = (V, E(\bar{x}))$ the partial graph of $G = (V, E)$ induced by the edges $e \in E$ having $\bar{x}_e > 0$.

Lemma 4 (a) *Suppose that the point* $\bar{x} \in \mathbb{R}^E$ *satisfies* (1). *Then the vertex set of every connected component of* $G(\bar{x})$ *defines a violated subtour elimination constraint if the graph* $G(\bar{x})$ *is disconnected.*
(b) *Suppose that the point* $\bar{x} \in \mathbb{R}^E$ *does not satisfy* (1). *There exists a nonempty set of* $W \subseteq V$ *such that* $\bar{x}(E(W)) > |W| - 1$, *if and only if there exists a nonempty set* W^*, *where* W^* *is contained in a connected component of* $G(\bar{x})$, *such that* $\bar{x}(E(W^*)) > |W^*| - 1$.

Thus we can assume that $G(\bar{x})$ is connected. Let $e \in E$ be any edge of $G(\bar{x})$ such that $\bar{x}_e \geqslant 1$. (We note that if $\bar{x}_e > 1$ holds for an edge $e = \{i, j\}$, then the set $W = \{i, j\}$ solves the subtour problem.) Clearly, edges with value 1 or bigger play a special role because of the form of the constraints $x(E(W)) \leqslant |W| - 1$. Indeed, we can shrink such edges away, thus reducing the size of the hard part of the subtour problem. To do so we carry out the following steps in sequence.

Shrinking an edge $e = \{i, j\}$ *of* $G(\bar{x})$

Step 1. Replace vertices i and j by a single vertex i_1.
Step 2. Every pair of edges $e = \{i, k\}$ and $e' = \{j, k\}$ is replaced by a single edge $e^* = \{i_1, k\}$ with weight $\bar{x}'_{e^*} = \bar{x}_e + \bar{x}_{e'}$.
Step 3. All other edges $\{i, l\}$ and $\{j, h\}$ of G, respectively, are replaced by the edges $\{i_1, l\}$ and $\{i_1, h\}$, respectively, and keep their old weights.

Denote by $G' = (V', E')$ the graph obtained from $G = (V, E)$ by shrinking edge e. Let $\bar{x}' \in \mathbb{R}^{E'}$ be the vector of weights that results from steps 2 and 3, and call $G(\bar{x}') = (V', E'(\bar{x}'))$ the partial graph induced by the edges $e \in E'$ having $x'_e > 0$.

It follows that $G(\bar{x}')$ is connected if $G(\bar{x})$ is connected. More importantly, one can prove the following lemma.

Lemma 5 *There exists* $W \subseteq V$, $W \neq \{i, j\}$, *such that* $\bar{x}(E(W)) > |W| - 1$ *holds, if and only if there exists a set* $W^* \subseteq V'$ *such that* $\bar{x}'(E'(W^*)) > |W^*| - 1$ *holds when an edge* $e = \{i, j\}$ *with* $\bar{x}_e \geqslant 1$ *is shrunk.*

Thus the shrinking operation suggests the following reduction procedure for the solution of the subtour problem.

A heuristic for the identification of subtour elimination constraints

Step 1. Given $\bar{x} \in \mathbb{R}^E$, $\bar{x} \geqslant 0$, set up the graph $G(\bar{x}) = (V, E(\bar{x}))$.
Step 2. If $\bar{x}_e < 1$ holds for all $e \in E(\bar{x})$ or if $|V| = 1$ holds, stop. If for some $e \in E(\bar{x})$, $\bar{x}_e > 1$ holds, stop: The set of original vertices corresponding to the vertices of edge e defines a set W such that $x(E(W)) > |W| - 1$ holds. Else let $e \in E(\bar{x})$ be such that $\bar{x}_e = 1$.
Step 3. Shrink edge e to get a new weight vector $\bar{x}'$ and the graph $G(\bar{x}')$. Replace $\bar{x}$ by $\bar{x}'$, $G(\bar{x})$ by $G(\bar{x}')$, and go to Step 2.

Using Lemma 5 one proves readily the following properties of the shrinking heuristic.

Lemma 6 *Given a graph $G=(V, E)$ and a point $\bar{x}\in\mathbb{R}^E$ satisfying $0\leqq\bar{x}_e\leqq 1$ for all $e\in E$, denote by $G^0=(V^0, E^0)$ the graph obtained by applying the shrinking heuristic to G and let x^0 be the vector of the respective edge weights.*
(a) *If $|V^0|=1$, then there does not exist a nonempty subset $W\subseteq V$ such that $\bar{x}(E(W))>|W|-1$ holds.*
(b) *If $|V^0|=2$, then there does not exist a nonempty proper subset $W\subseteq V$ such that $\bar{x}(E(W))>|W|-1$ holds.*
(c) *If $|V^0|\geqq 3$ and there exists a nonempty (a nonempty and proper, respectively) subset $W\subseteq V$ such that $\bar{x}(E(W))>|W|-1$ holds, then there exists a nonempty (a nonempty and proper, respectively) subset $W^*\subseteq V^0$ such that $x^0(E^0(W^*))>|W^*|-1$. In other words, there exists a subtour elimination constraint violated by $\bar{x}$ in the original graph G if and only if there exists a subtour elimination constraint violated by x^0 in the reduced graph G^0.*

The shrinking heuristic for the identification of violated subtour elimination constraints has been considered by Crowder & Padberg [1980] and was found to be very effective. A similar procedure was used by Land [1979]. Moreover, if the calculations of this algorithm are carefully implemented, then the algorithm requires a total computational effort which is polynomially bounded. We leave the calculation of the polynomial bound as an exercise to the reader. If several violated subtour elimination constraints are found, then it is advisable to generate all of them except, of course, equivalent ones. To minimize storage requirements one should store the subtour elimination constraint on $V-W$ rather than the one on W if $|W|>\lceil n/2\rceil$.

We note for completeness that Padberg & Hong [1980] have given two further heuristics which (sometimes) solve the *primal* facet identification problem when restricted to subtour elimination constraints. The procedures are simple to program and have been found effective in actual computation. However, they are geared to the primal cutting plane approach to the TSP, and the reader is referred to Padberg & Hong [1980, Section 4.1] for details. One verifies readily that there are at most polynomially many possible subtour constraints for the primal facet identification problem, and thus one can effectively enumerate all possibilities.

2.2 A polynomial algorithm for subtour elimination constraints

The shrinking heuristic does not always solve the subtour problem. Because of the properties described in Lemma 6(c), however, the heuristic can always be carried out prior to the execution of an exact algorithm. It should always be carried out prior to any further calculation because of the reduction in problem size that is generally achieved.

The solution of the subtour problem is more difficult than it may appear.

To give a complete answer we first treat the case where $\bar{x}\in\mathbb{R}^E$ satisfies (1) and then discuss the general case of arbitrary $\bar{x}\in\mathbb{R}^E$, because it is of interest if we want to optimize over the 1-tree polytope Q^n_{1T} or its monotonization $\tilde{Q}^n_{1T}$ introduced in Section 3.1 of Chapter 8. In both cases, however, we may assume that $\bar{x}\in\mathbb{R}^E$ satisfies $0\leqq\bar{x}_e<1$ for all $e\in E$, i.e. that the heuristic has been applied already. If $\bar{x}\in\mathbb{R}^E$ satisfies (1), then by summing over all vertices in some set $W\subseteq V$ we get

$$2\bar{x}(E(W))+\bar{x}(\{W: V-W\})=2|W|$$

where

$$\{W: V-W\}=\{e\in E\mid e \text{ has exactly one end in } W\}$$

is the *cut-set* defined by W. The expression $\bar{x}(\{W: V-W\})$ is the *capacity* of the cut-set and we abbreviate it by $\bar{x}(W: V-W)$. One can show that the subtour problem is equivalent to the following cut problem.

Cut problem *Given $\bar{x}\in\mathbb{R}^E$ satisfying $0\leqq\bar{x}_e\leqq 1$ for all $e\in E$ and the equations (1), find a nonempty proper subset W of V such that $\bar{x}(W: V-W)<2$ holds, or prove that no such $W\subseteq V$ exists.*

Consequently, what we are interested in is finding a *minimum capacity cut-set* in the graph $G(\bar{x})$ where the capacities are given by the weights $\bar{x}_e$, $e\in E$. If the minimum cut-set in $G(\bar{x})$ has a capacity which is greater than or equal to 2, then we conclude that there exists no subtour elimination constraint that is violated by $\bar{x}$. Else a vertex set W given by a minimum capacity cut-set defines a violated subtour elimination constraint. The determination of a minimum capacity cut-set in an undirected graph can be carried out by an algorithm given by Gomory & Hu [1961] which we discuss next.

Let $u\neq v\in V$ be any two vertices of G and consider the problem of finding a minimum cut-set:

$$\min\{\bar{x}(W: V-W)\mid W\subseteq V, u\in W, v\in V-W\}$$

separating vertices u and v or, for short, the problem of finding a minimum (u, v)-cut in G. Due to the famous maximum flow–minimum cut theorem (see e.g. Ford & Fulkerson [1962] or Lawler [1976]), the calculation of a minimum (u, v)-cut can be carried out by calculating the maximum flow from u to v (or from v to u, as the underlying graph is symmetric). Thus, considering all possible pairs of vertices of G, we can find the *minimum minimorum* among all cuts $\{W: V-W\}$ satisfying $W\neq\emptyset$ and $V-W\neq\emptyset$ (i.e. a minimum cut-set in the graph G) with $\frac{1}{2}|V|(|V|-1)$ maximum flow calculations. The Gomory & Hu algorithm reduces this computation to at most $|V|-1$ maximum flow calculations.

We state their algorithm in a somewhat more general form which is due to Hu [1969, p. 139] to solve the problem

$$\min\{\min\{\bar{x}(W: V-W)\mid W\subseteq V, u\in W, v\in V-W\}\mid u\neq v, u\in N, v\in N\}$$

where $N \subseteq V$ is a set of specified vertices of V called *terminal vertices* of the graph G.

Minimum cut algorithm

Input: An undirected connected graph $G = (V, E)$ with capacities $\bar{x}_e \geq 0$ for all $e \in E$ and a subset $N \subseteq V$ of terminal vertices.

Step 0. (*Initialize*) $L = \{0\}$, $G^0 = G$, $N^0 = N$, G_T has one vertex corresponding to G^0.

Step 1. If $L = \emptyset$, stop. Else let $l = \min\{j \mid j \in L\}$, $t = \max\{j \mid j \in L\}$.

Step 2. If $|N^l| \leq 1$, replace L by $L - \{l\}$ and go to Step 1. Else choose $u \neq v$, $u \in N^l$, $v \in N^l$.

Step 3. Calculate a minimum capacity (u, v)-cut-set $\{M : V^l - M\}$ using a maximum flow algorithm (V^l is the vertex set of G^l) and let v^l be its capacity.

Step 4. Create a new graph G^{t+1} (and a new graph G^{t+2}, respectively) by shrinking all edges of G^l in $V^l - M$ (in M, respectively) using the procedure given above, thus getting a new pseudo-vertex in each new graph. Let N^{t+1} (N^{t+2}, respectively) be the terminal vertices that G^{t+1} (G^{t+2}, respectively) inherit from G^l excepting those contained in the pseudo-vertices. Replace L by $L \cup \{t+1, t+2\} - \{l\}$

Step 5. Split the vertex of G_T corresponding to G^l into two vertices corresponding to G^{t+1} and G^{t+2}, respectively, and connect them by an edge with weight v^l. Connect a former neighbor G^l of G^l to G^{t+1} if $N^l \subseteq M$ (where M is considered as a subset of V), to G^{t+2} otherwise, and keep the old edge weights. Go to Step 1.

The output of the algorithm is a tree G_T whose edges have weights corresponding to certain (minimum) cut-sets in G and whose vertices correspond to subsets of the original graph G containing exactly one terminal vertex. The graph G_T is called the *cut tree* of G.

While Exercise 15 captures the more obvious aspects of the algorithm the next theorem, from Hu [1969, p. 138], asserts far more.

Theorem 4 *The capacity of a minimum (u, v)-cut separating any two terminal vertices u and v of the graph G is equal to the minimum of the weights of the edges in the unique path of the cut tree G_T connecting the two vertices of G_T containing u and v.*

Picking any edge of G_T with minimum weight and deleting it from G_T, we get two subtrees G_T^1 and G_T^2 of G_T. Let M be the set of vertices of G that correspond to the vertices of G_T^1. Then by Theorem 4 the cut-set $\{M : V - M\}$ is a cut-set which is minimum among all cut-sets separating any two terminal vertices of G and thus, in particular, a minimum cut-set of G if $N = V$ holds.

We consider now the case where the point $\bar{x} \in \mathbb{R}^E$ of the subtour problem does not satisfy the equations (1). More precisely, we are interested in deciding whether or not $\bar{x}$ satisfies the system of inequalities

$$\begin{aligned} x(E(W)) &\leq |W| - 1 \quad \text{for all } \emptyset \neq W \subseteq V, \\ x &\geq 0, \end{aligned} \tag{5}$$

which is known to have as extreme points the incidence vectors of trees and forests of the graph G [Edmonds, 1970]. The inequalities (5) occur in the definition of the 1-tree polytope Q_{1T} and since there are exponentially many constraints in (5), the solution of the subtour problem in this case is by no means obvious; nor does it follow from the previous arguments. Since we no longer have the constraints (1), we cannot transform the problem into the cut problem. However, as Lemma 4(b) and Lemma 6 show, the application of the shrinking heuristic is justified in this case as well, thereby reducing the problem size.

To solve the general case of the subtour problem we define a capacitated network G^* using $G = (V, E)$ and the weights (capacities) $\bar{x}_e$, $e \in E$, as follows.

Construction of the capacitated network G^*

Step 1. Every (undirected) edge $e = \{u, v\}$ of G is replaced by a pair of arcs, (u, v) and (v, u) with capacities $c_{uv} = c_{vu} = \frac{1}{2}\bar{x}_e$.

Step 2. A source labeled 0 with arcs $(0, v)$ and capacities $c_{0v} = \max\{\frac{1}{2}b_v - 1, 0\}$ and a sink labeled $n+1$ with arcs $(v, n+1)$ and capacities $c_{v,n+1} = \max\{1 - \frac{1}{2}b_v, 0\}$ are adjoined to G, where $b_v = \bar{x}(\delta(v))$ for all $v \in V$ and $n = |V|$.

Now one calculates

$$c(W \cup \{0\} : V \cup \{n+1\} - W) = |W| - \bar{x}(E(W)) + \sum_{v \in V} \max\{\tfrac{1}{2}b_v - 1, 0\},$$

and since the last term in the equation is independent of W we have proven the following result [Padberg & Wolsey, 1983].

Theorem 5 *The minimization problem* $\min\{|W| - \bar{x}(E(W)) \mid W \subseteq V, W \neq \emptyset\}$ *is equivalent to the problem of finding a minimum $(0, n+1)$-cut in G^* satisfying $W \neq \emptyset$, i.e. to the problem*

$$\min\{c(W \cup \{0\} : V \cup \{n+1\} - W) \mid W \subseteq V, W \neq \emptyset\}. \tag{6}$$

Note that in problem (6) we do not require that W be a proper subset of V, as was the case in the cut problem. However, the restriction that $W \neq \emptyset$ implies that a single maximum-flow computation does not suffice. To satisfy the condition $W \neq \emptyset$, we solve $n-2$ maximum flow problems with capacities as defined before except that for the kth problem we set $c_{0,k} = +\infty$, and if $k \geq 2$ holds, we set $c_{v,n+1} = +\infty$ for $v = 1, \ldots, k-1$, where k assumes the

values $1, 2, \ldots, n-2$. We record the cut-set that is minimum over the $n-2$ iterations; the resulting vertex set W solves the subtour problem without the restriction $|W| \leqq n-1$ in the general case.

It is perhaps worthwhile to note that the construction underlying Theorem 5 can be modified to suit also the case where the point $\bar{x} \in \mathbb{R}^E$ satisfies the equations (1) and where we are interested in satisfying the restriction that W be a proper subset of V. In this case the minimum of (6) is obviously 0. To accommodate the condition $W \neq V$, since (1) holds, we can simply define the arc capacity $c_{n,n+1}$ to be $+\infty$ in all of the $n-2$ maximum flow calculations and proceed as previously.

Thus, we have given two procedures for solving the subtour problem but the second one, while applicable to a more general situation than is the case of the TSP, requires the same computational effort as the first one. The minimum cut algorithm will be used again in the next section. Because the maximum flow calculation can be carried out in $O(n^3)$ steps we have proven the following result.

Theorem 6 *The identification of subtour elimination constraints can be carried out by a polynomial algorithm requiring at most $O(n^4)$ steps.*

Note that Theorem 6 (together with the ellipsoid method) implies that we can optimize in polynomial time over the 1-tree polytope Q_T^n and the monotone 1-tree polytope $\tilde{Q}_T^n$ (see (12) to (18) of Chapter 8). (Of course this can be done with the greedy algorithm in a much simpler way).

Moreover, using Exercise 9 we can conclude that the identification problem for subtour elimination constraints can be carried out in polynomial time for the asymmetric case as well. So Theorem 6, Exercise 9 and the ellipsoid method yield polynomial algorithms for optimizing linear objective functions over the branching polytope $\tilde{P}_B^n$, the antibranching polytope $\tilde{P}_B^n$, the arborescence polytope P_B^n, and the antiarborescence polytope P_B^n (see (26) to (31) of Chapter 8); in other words, we obtain new polynomial alternatives to the algorithm in Edmonds [1967].

Exercises

8. Prove Lemma 3.
9. Let $a^T x \leqq a_0$ be an inequality valid for P_T^n with $a_{ij} = a_{ji}$, $1 \leqq i, j \leqq n$, and let $b \in \mathbb{R}^E$ be the vector defined by $b_{ij} = a_{ij}$, $1 \leqq i, j \leqq n$. Then $b^T x \leqq a_0$ is valid for Q_T^n. Moreover, if $\bar{x} \in \mathbb{R}^A$ and $\bar{\bar{x}} = f(\bar{x}) \in \mathbb{R}^E$ is the symmetrized version of $\bar{x}$, then $\bar{x}$ violates $a^T x \leqq a_0$ if and only if $\bar{\bar{x}}$ violates $b^T x \leqq a_0$.
10. Prove Lemma 4.
11. Prove Lemma 5.
12. Apply the shrinking heuristic to the graph of Figure 9.1.
13. Prove Lemma 6.
14. (a) Apply the shrinking heuristic to the graph $G(\bar{x})$ of Figure 9.2.

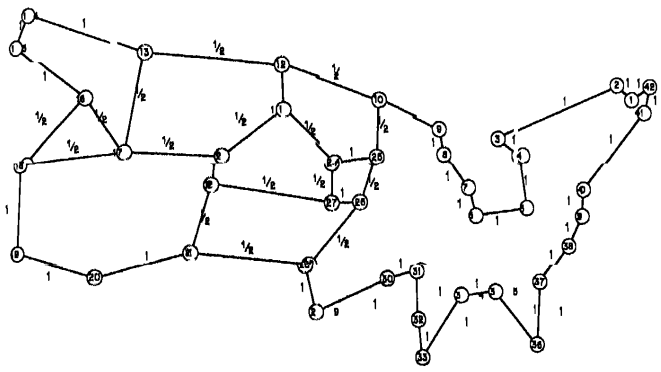

Figure 9.1 Fractional solution of the 42-city problem by Dantzig, Fulkerson & Johnson [1954]

(b) Find a violated subtour elimination constraint in the reduced graph by inspection.

15. (a) Prove that Step 3 of the minimum cut algorithm is carried out exactly $|N| - 1$ times.

(b) Prove that the graph G_T constructed by the minimum cut algorithm is a tree.

16. Apply the minimum cut algorithm to the graph obtained from Exercise 14(a).

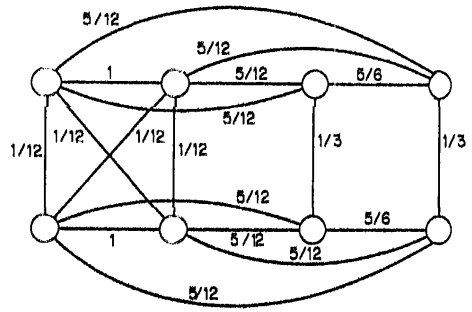

Figure 9.2 Graph for Exercise 14

3 IDENTIFICATION OF CLIQUE TREE INEQUALITIES

For the case of the general clique tree inequalities, the facet identification problem is not solved to date nor do we know any (nontrivial) heuristic for it. (For an introduction to clique tree inequalities see Section 4.2 of Chapter 8.) However, the problem of identifying any violated 2-matching constraint has been completely solved, and there is a reasonable heuristic for comb constraints. When restricted to comb constraints, the facet identification problem can be formulated in the symmetric case as follows, where $G = (V, E)$ is the complete graph having n vertices.

Comb problem *Given a point* $\bar{x} \in \mathbb{R}^E$ *satisfying* (1), *find sets* $H, T_1, \ldots, T_k \subseteq V$ *with the properties*
(a) $|H \cap T_i| \geq 1, \quad i = 1, \ldots, k,$
(b) $|T_i - H| \geq 1, \quad i = 1, \ldots, k,$
(c) $T_i \cap T_j = \emptyset, \quad 1 \leq i < j \leq k,$
(d) $k \geq 1$ *and* k *odd,*
such that $\bar{x}(E(H)) + \sum_{i=1}^{k} \bar{x}(E(T_i)) > |H| + \sum_{i=1}^{k} (|T_i| - 1) - \lceil k/2 \rceil$; *or prove that no such sets* $H, T_1, \ldots, T_k \subseteq V$ *exist.*

As in Chapter 8, we call a collection of subsets $H, T_1, \ldots, T_k \subseteq V$ satisfying properties (a), (b), (c), (d) a *comb* in G and abbreviate it by its edge set $C = E(H) \cup \bigcup_{i=1}^{k} E(T_i)$. The quantity

$$s(C) := |H| + \sum_{i=1}^{k} (|T_i| - 1) - \left\lceil \frac{k}{2} \right\rceil$$

is the *size* of the comb C. By slight abuse of notation we shall write

$$x(C) \leqslant s(C)$$

to denote a comb inequality. Note that in contrast to the definition given in Chapter 8, we permit here $k = 1$. Combs with $k = 1$ are dominated by subtour elimination constraints (cf. Exercise 14(a) of Chapter 8 which carries over to combs as well). However, for identification purposes it is more convenient initially to permit $k = 1$. As we are interested in finding undominated combs, we will then have to discuss a clean-up procedure which will be discussed below.

Comb inequalities have coefficients of 0, 1 and 2. The smallest graph admitting a comb having a coefficient of 2 (as well as 0's and 1's) in the associated inequality that is not equivalent to a simple comb, has $n = 8$ vertices and was found in trying to eliminate the point $\bar{x}$ shown in Figure 9.3. (There exists a comb inequality which is violated by $\bar{x}$, and the reader is asked to try to find it prior to reading the material of Section 3.3.)

To accommodate the case of the asymmetric TSP, we proceed as in the case of subtour elimination constraints (cf. Exercise 9).

Lemma 7 *Let* $\bar{x} \in \mathbb{R}^E$ *satisfy* (1) *and let* $x^* \in \mathbb{R}^E$ *be the incidence vector of a tour, i.e.* x^* *is an extreme point of* Q_1^n. *Then there exists a comb* C *such that*

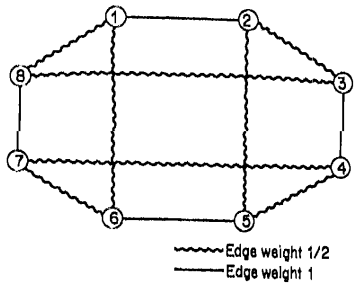

Figure 9.3 Graph illustrating comb constraints

$x^*(C) = s(C)$ *and* $\bar{x}(C) > s(C)$ *hold if and only if there exists a comb* C *such that* $\bar{x}(C) > s(C)$ *holds where* $\bar{x} = (n-1/n)x^* + (1/n)\bar{x}$.

Thus the primal facet identification problem, when restricted to comb constraints, can be reduced to the comb problem.

3.1 A polynomial algorithm for 2-matching constraints

In the case of 2-matching constraints we have equality in properties (a) and (b) and therefore the problem can be reformulated as follows:

2-Matching problem *Given a point* $\bar{x} \in \mathbb{R}^E$ *satisfying* (1), *find a set* $H \subseteq V$ *and a set* T *of edges* $e_1, \ldots, e_{|T|}$ *in* E *with the properties*
(a) e_i *has one endpoint in* H, $i = 1, \ldots, |T|$,
(b) $e_i \cap e_j = \emptyset, \quad 1 \leqslant i < j \leqslant |T|$,
(c) $|T|$ *odd,*
such that $\bar{x}(E(H)) + \bar{x}(T) \geq |H| + \lfloor \frac{1}{2}|T| \rfloor$; *or prove that no such sets* $H \subseteq V$ *and* $T \subseteq E$ *exist.*

Formulated this way, the problem looks rather difficult to solve, and in order to tackle it we will make use of the assumption that the point $\bar{x} \in \mathbb{R}^E$ satisfies (1). Indeed, unlike the case of the subtour elimination constraints, we do not know how to solve the above problem in the general case.

Remark 1 We note that the 2-matching problem for a point $\bar{x} \in \mathbb{R}^E$ satisfying $Ax \leqslant 2$ rather than $Ax = 2$ in (1) can be reduced to the equality case by introducing a dummy vertex to the graph $G = (V, E)$ to absorb the slack in the inequalities [Padberg & Rao, 1982].

Introducing slack variables for the upper bounds of (1), we can write the

system (1) as follows:

$$x(\delta(v)) = 2 \quad \text{for all } v \in V, \tag{7}$$

$$x_e + t_e = 1 \quad \text{for all } e \in E, \tag{8}$$

$$x_e \geqslant 0, \quad t_e \geqslant 0 \text{ for all } e \in E.$$

Adding (7) over all $v \in H$ and (8) over all $e \in T$ and adding together yields

$$2x(E(H)) + x(H : V-H) + x(T) + t(T) = 2|H| + |T|,$$

and consequently

$$2(x(E(H)) + x(T)) = 2|H| + |T| - x(H : V-H) - t(T) + x(T).$$

It follows that

$$\bar{x}(E(H)) + \bar{x}(T) > |H| + \lfloor \tfrac{1}{2}|T| \rfloor \tag{9}$$

holds if and only if

$$\bar{x}(H : V-H) - \bar{x}(T) + \sum_{e \in T} (1 - \bar{x}_e) < 1 \tag{10}$$

holds. Now $T \subseteq \{H : V-H\}$ holds and thus *all* edges of the cut-set $\{H : V-H\}$ appear in (10) *exactly once*, except that the edges in T appear as variables that are complemented into their upper bounds. As in the case of the subtour elimination constraints, when the 2-matching constraints are brought into the form (10), the comb problem reduces to some kind of minimum cut problem with the additional feature that for some odd subset of its edges we need to complement the variables.

To find suitable sets H and T, we recall a standard device to transform a 2-matching problem in 0–1 variables into a matching problem without upper bounds on a larger graph with two classes of vertices labeled 1 and 2. Split each edge of G into three edges by inserting into an edge $e = \{i, j\}$ two vertices i_1 and i_2, with vertex label 1, and define $\bar{x}_{i,i_1} = \bar{x}_{i_2,j} = \bar{x}_e$ and $\bar{x}_{i_1,i_2} = 1 - \bar{x}_e$ to be the new weights. Doing so for all edges, one gets a new vector $\bar{x}$ of weights and a new graph G' with a vertex set $V' = V \cup V_*$ such that

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

$$\bar{x}(\delta(v)) = 1 \quad \text{for all } v \in V_*.$$

Clearly, G' has an even number of vertices with vertex label 1 and some other number of vertices (the original ones) with vertex label 2. Call the vertices with a vertex label 1 *odd* vertices and the other ones *even* vertices. By making the same transformation that brought us to (10) and ignoring for the moment the requirement (b) of the 2-matching problem, it follows that we want to find in G' a cut-set containing an odd number of odd vertices and having a capacity of less than 1.

Rather than splitting each edge into three edges, it suffices to split each edge into two edges, and this is important, since we want to keep the problem size as small as possible.

Labeling procedure constructing $G^*(\bar{x})$ from $G(\bar{x})$

Input: The graph $G(\bar{x})$ as defined in Section 2.1. All vertices of $G(\bar{x})$ are labeled even.

Step 1. Pick any edge $e \in E(\bar{x})$.

Step 2. Let $e = \{u, v\}$ and replace e by two edges $\{u, i_e\}$ and $\{i_e, v\}$ where i_e is a new vertex. Vertex i_e is labeled odd. Vertex u gets a new label which is odd if its old label is even and which is even if its old label is odd. Vertex v retains its old label. The edge $\{u, i_e\}$ gets the weight $1 - \bar{x}_e$ and edge $\{i_e, v\}$ gets the weight $\bar{x}_e$. Mark edge e as having been scanned.

Step 3. If all edges of $G(\bar{x})$ have been scanned, stop. Else pick any unscanned edge $e \in E(\bar{x})$ and go to Step 2.

As we shall see later, the number of vertices labeled odd in the above procedure determines the number of times a maximum flow algorithm has to be run in order to check whether all 2-matching constraints are satisfied. Thus, it is important to pick edges in Step 3 and their endpoints in Step 2 in such a way that the final number of vertices labeled odd is a minimum. In fact, there are (fast) labeling techniques which guarantee that for a connected graph $G(\bar{x})$ the number of old vertices in $G^*(\bar{x})$ (i.e. those from $G(\bar{x})$) labeled odd is zero or one. The latter case necessarily occurs if the number of edges of $G(\bar{x})$ is odd.

Denote by $\bar{V} = V \cup V_E$ the vertex set of $G^*(\bar{x})$, where $V_E = \{i_e \mid e \in E(\bar{x})\}$ is the set of newly introduced vertices. Note that to every edge $e \in E(\bar{x})$ there corresponds a *unique* new vertex $i_e \in V_E$. Denote by F the edge set of $G^*(\bar{x})$ and let $\bar{y}_f$ for $f \in F$ be the weight (or the *capacity*) of edge f. We claim that $G^*(\bar{x})$ has an even number of vertices which are labeled odd, and leave the proof to the reader.

A cut-set $\{U : \bar{V} - U\}$ of $G^*(\bar{x})$ is called *odd* if U contains an odd number of odd labeled vertices (and any number of even labeled vertices). A *minimum odd cut-set* is an odd cut-set having a minimum capacity among all odd cut-sets.

Lemma 8 *Let $H \subseteq V$ and $T \subseteq \{H : V-H\}$ be such that $H \neq \emptyset$ and $|T|$ is odd. Define*

$$H_1 = \{i_e \in V_E \mid e \in E(H), e \in E(\bar{x})\},$$

$$U_1 = \{i_e \in V_E \mid e \in \{H : V-H\}, e \in E(\bar{x})\}.$$

Partition U_1 into four disjoint sets:

$$U_{11} = \{i_e \in U_1 \mid e \notin T, \bar{y}_{ki_e} = 1 - \bar{x}_e \text{ for some } k \in H\},$$

$$U_{12} = \{i_e \in U_1 \mid e \notin T, \bar{y}_{ki_e} = \bar{x}_e \text{ for some } k \in H\},$$

$$U_{13} = \{i_e \in U_1 \mid e \in T, \bar{y}_{ki_e} = 1 - \bar{x}_e \text{ for some } k \in H\},$$

$$U_{14} = \{i_e \in U_1 \mid e \in T, \bar{y}_{ki_e} = \bar{x}_e \text{ for some } k \in H\},$$

and define $U = H \cup H_1 \cup U_{11} \cup U_{14}$. *Then*
(a) U *has an odd number of odd labeled vertices*;
(b) $\bar{y}(U : \bar{V} - U) = \bar{x}(H : V - H) - \bar{x}(T) + \sum_{e \in T}(1 - \bar{x}_e)$.

Thus we know how to construct an odd cut-set in $G^*(\bar{x})$ from a pair of sets $H \subseteq V$ and $T \subseteq \{H : V - H\}$ that define a 2-matching constraint.

The next theorem, which is given in a more general form by Padberg & Rao [1982], gives an equivalent formulation of the comb problem when one temporarily ignores condition (b) of the 2-matching problem.

Theorem 7 *Given a point* $\bar{x} \in \mathbb{R}^E$ *satisfying* (1) *there exist* $H \subseteq V$ *and* $T \subseteq \{H : V - H\}$ *with* $|T|$ *odd such that* (9) *holds if and only if the capacity of a minimum odd cut-set in* $G^*(\bar{x})$ *is less than* 1. *Furthermore, if* $\{U : \bar{V} - U\}$ *is a minimum odd cut-set with capacity less than* 1, *then* $H = U \cap V$ *and* T, *where*

$$T = \{e \in E(\bar{x}) \mid \exists f \in \{U : \bar{V} - U\} \text{ such that } f = \{i_e, j\} \text{ for some } j \in V \text{ and } \bar{y}_f = 1 - \bar{x}_e\}, \tag{11}$$

satisfy (9) *and* $|T|$ *is an odd number.*

Before stating an algorithm to solve the minimum odd cut-set problem in the labeled graph $G^*(\bar{x})$, we first show how we can alter a minimum odd cut-set of capacity less than 1 so as to satisfy the above mentioned property (b). If $|T| = 1$ holds for the set T defined in (11), then (b) is satisfied and the comb problem is solved.

Remark 2 By Remark 1(b) at the end of Section 4.3 in Chapter 8, a 2-matching constraint with $|T| = 1$ is dominated by a subtour elimination constraint with respect to Q^n_T. Thus if the calculation of a minimum odd cut-set yields a cut with capacity less than 1 and a set T with $|T| = 1$ we find a violated subtour elimination constraint which in numerical calculations for TSPs should be stored, rather than the dominated 2-matching constraint, because in this case the latter does not define a facet.

Thus we can assume without loss of generality that $|T| \geq 3$ holds. One establishes the following claim.

Lemma 9 *If the capacity of a minimum odd cut-set in* $G^*(\bar{x})$ *is less than* 1 *and* $|T| \geq 3$ *holds for the pair* H, T *defined in Theorem* 7, *then* $|H| \geq 3$ *holds.*

Consequently, to satisfy property (b) of the 2-matching conditions we can assume that both $|H| \geq 3$ and $|T| \geq 3$ hold in Theorem 7. Suppose now without loss of generality that $e_k \cap e_{k-1} \neq \emptyset$ where $k = |T|$. Let $e_k = \{u, v\}$ and $e_{k-1} = \{u, w\}$. If $u \notin H$, we define $H' = H \cup \{u, v, w\}$ and $T' = T - \{e_k, e_{k-1}\}$. If $u \in H$, we define $H' = H - \{u\}$ and T' as before.

Lemma 10 *Suppose the capacity of a minimum odd cut-set in* $G^*(\bar{x})$ *is less than* 1 *and that the pair* H, T *obtained from Theorem* 7 *violates property* (b).

Then the pair H', T' *constructed above defines a minimum odd cut-set in* $G^*(\bar{x})$ *via the construction of Lemma* 8.

If the pair H', T' thus obtained continues to violate (b) we reapply the above construction and in at most $\lfloor \frac{1}{2}|T| \rfloor$ steps we stop with a pair H', T' such that all of the 2-matching properties (a), (b), (c) and (9) hold. The exercise of satisfying (b) is necessary since we are interested in generating *facets* of Q^n_T rather than valid inequalities.

As the above construction shows, the process of converting a minimum odd cut-set in $G^*(\bar{x})$ to a pair H, T defining a facet-inducing linear inequality for Q^n_T can be done in polynomial time. The question is whether we can find a minimum odd cut-set in $G^*(\bar{x})$. This is indeed the case and can be done by the following modification from Padberg & Rao [1982] of the minimum cut algorithm given by Gomory & Hu [1961].

Minimum odd cut algorithm

Input: An undirected connected graph $G = (V, E)$ with capacities $\bar{x}_e \geq 0$ for all $e \in E$ and a subset $N \subseteq V$ of vertices labeled odd having an even cardinality $|N|$.

Step 1. Using the minimum cut algorithm with N being the terminal vertices, compute the cut tree $G_T = (N_T, F_T)$ and let d_f be the weight of edge $f \in F_T$. Initialize $L = \{0\}$ and $G^0_T = G_T$, $N^0_T = N_T$, $F^0_T = F_T$, $\bar{c} = +\infty$.

Step 2. If $L = \emptyset$, stop. Else let $l = \min\{j \mid j \in L\}$, $t = \max\{j \mid j \in L\}$.

Step 3. Find $h \in F^l_T$ such that

$$d_h = \min\{d_f \mid f \in F^l_T\}$$

holds and denote by $G^{t+i}_T = (N^{t+i}_T, F^{t+i}_T)$ the two subtrees obtained from G^l_T by removing edge h from G^l_T where $i = 1, 2$.

Step 3.1. If $|N^{t+1}_T|$ is odd and $d_h < \bar{c}$, replace $\bar{c}$ by d_h, replace L by $L - \{l\}$; set $f^* = h$ and go to Step 2.

Step 3.2. If $|N^{t+1}_T|$ is odd and $d_h \geq \bar{c}$, replace L by $L - \{l\}$ and go to Step 2.

Step 3.3. If $|N^{t+1}_T|$ is even, replace L by $L \cup \{t+1, t+2\} - \{l\}$ and go to Step 2.

The output of the algorithm is an edge f^* of the cut tree G_T having the property that it is an edge with smallest weight among those edges whose removal decomposes G_T into two subtrees each having an odd number of odd labeled vertices. Thus the vertex set of any one of the two subtrees of G_T found in this way defines the vertex set of a minimum odd cut-set in G. Furthermore, the value of $\bar{c}$ at the end of the calculation is the capacity of a minimum odd cut set. The algorithm iterates at most $|N_T| - 1$ times and requires one minimum cut calculation.

Theorem 8 *The identification of 2-matching constraints can be carried out by a polynomial algorithm requiring at most* $O((n+m)^4)$ *steps, where* $n = |V|$ *and* $m = |E(\bar{x})|$.

Proof The set-up of the labeled graph $G^*(\bar{x})$ requires the scanning of each edge of the graph $G(\bar{x})$ and can be done in time linear in $m = |E(\bar{x})|$. The 'cleaning up' of the pair H, T obtained by Theorem 7 to satisfy condition (b) of the 2-matching problem can be done in time linear in $n = |V|$. The execution of the minimum odd cut algorithm requires at most $n+m$ steps since $G^*(\bar{x})$ has at most $n+m$ odd vertices. Thus the entire computation is dominated by the calculation of the cut tree G_T using the minimum cut algorithm in Step 1 above, and thus the claim is proved. □

From Theorems 5 and 7, the following result now follows by the same line of reasoning that preceded Theorem 3.

Theorem 9 *There exists a polynomial algorithm to solve the optimization problem*

$$\min\{c^T x \mid x \in Q^n_{1T} \cap Q^n_{2M}\},$$

where Q^n_{1T} is the 1-tree polytope and Q^n_{2M} is the perfect 2-matching polytope (cf. Section 3 of Chapter 8).

By Remark 1, it follows that Theorem 9 remains true if we replace Q^n_{1T} and Q^n_{2M} by their monotonizations $\tilde{Q}^n_{1T}$ and $\tilde{Q}^n_{2M}$, respectively. Since the traveling salesman polytope Q^n_T is contained in the intersection of the 1-tree polytope with the perfect 2-matching polytope, we are thus guaranteed to find better lower bounds on the optimal tour length in polynomial time than was previously the case when the 1-tree relaxation and the 2-matching relaxation (but not the intersection of the two) could be solved in polynomial time.

As a recommendation for the implementation of the algorithm, we note that *all* odd cut-sets with capacity less than 1 in $G^*(\bar{x})$ define violated 2-matching constraints. Thus having calculated the cut-tree G_T for $G^*(\bar{x})$, one should generate and store *all* nonequivalent 2-matching constraints that are obtained by breaking the cut tree into two subtrees each having an odd number of vertices if an edge of G_T with weight less than 1 is removed. To minimize space one stores, of course, the smaller one of the two equivalent ones that result from the removal of a single edge.

3.2 Heuristics for 2-matching constraints

The reader who *actually* carried out Exercise 18 will be relieved to learn that the *size* of the problem necessary to solve the comb problem can generally be reduced substantially. In this section we state two procedures, one of which is in the spirit of the shrinking heuristic for the subtour elimination constraints which should *always* be applied before using the exact algorithm. The second one is an easy-to-implement straight heuristic (no claim to generality), which, however, was used with success in the computational study by Padberg & Hong [1980].

Let $G(\bar{x})$ be the graph defined in Section 2.1 and consider any path P connecting two vertices a and b such that every edge e of the path P has an weight $\bar{x}_e = 1$. Let a and b be such that both a and b are incident to edges which have fractional weights $0 < \bar{x}_e < 1$, i.e. the path P is a maximal path of edges having weights of 1. Let us call such a path a 1-*path* in $G(\bar{x})$. Now we delete all vertices of the 1-path except the two end vertices a and b and delete all edges of the path. We label the vertices a and b odd. We continue to do so until all 1-paths are removed. We call the graph that results from this construction G_1 and label all unlabeled vertices of G_1 even. Note that all edges of G_1 have weight less than 1.

Lemma 11 *There exists a minimum odd cut-set with capacity less than 1 in the labeled graph $G^*(\bar{x})$ if and only if there exists a minimum odd cut-set with capacity less than 1 in the graph G^*_1 obtained by applying the labeling procedure with the graph G_1 constructed above as input.*

Furthermore, it is clear how to construct from an odd cut-set with capacity less than 1 in G^*_1 an odd cut-set with capacity less than 1 in $G^*(\bar{x})$: if for any removed 1-path the respective two endpoints a and b are on the same side of the cut, then all vertices of the path (including the ones that correspond to the edges of the path) are put on the same side of the cut as the endpoints. If the endpoints a and b are on different sides of the cut, then all vertices of the path other than vertex a (including the ones that correspond to the edges of the path) are put on the same side of the cut as vertex b.

Consider next the graph G_1. An *alternating path* of vertices $v_1, \ldots, v_k$ of G_1 is a path having at least three vertices such that vertices v_i for $i = 2, \ldots, k-1$ are labeled odd in G_1. By construction, the weights of successive edges of an alternating path assume values α and $1-\alpha$, respectively, where $0 < \alpha < 1$. We permit the case $v_1 = v_k$ and v_1 or v_k may be labeled even or odd. As was the case with paths given by edges having weights of 1, we consider only alternating paths which are maximal with respect to their defining properties.

Shrinking alternating paths of the graph G_1

Input: The labeled graph G_1 constructed above and an alternating path of vertices $v_1, \ldots, v_k$.

Step 1. If $v_1 \neq v_k$, go to Step 2. Else shrink the cycle to a single vertex which retains the old label of v_1 if k is odd and which gets the opposite label to the old label of v_1 if k is even.

Step 2. Let $\alpha = \bar{x}_{v_1 v_2}$ be the weight of the first edge in the path and shrink the path with vertices $v_1, \ldots, v_k$ to a path having three vertices v_1, v'_2, v_k by introducing a new vertex v'_2 which is labeled odd. Define edge weights $\bar{x}_{v_1 v'_2} = \alpha$ and $\bar{x}_{v'_2 v_k} = 1-\alpha$. Vertex v_1 retains its old label. Vertex v_k retains its old label if k is odd. Otherwise v_k is labeled odd if its old label is even, and even otherwise.

The validation of the shrinking procedure is left to the reader. (*Hint:* Consider the graph that results if the labeling procedure is applied. Then argue the obvious simplifications that are possible if one looks for an odd cut-set with capacity less than 1.) We now can state a heuristic which uses the two shrinking operations described above.

A heuristic for 2-matching constraints

Input: The graph $G(\bar{x})$ defined in Section 2.1.

Step 1. Shrink all 1-paths in $G(\bar{x})$ to get the labeled graph G_1 described above.

Step 2. If no alternating paths in G_1 exist, stop. Else shrink all alternating paths in G_1 using the above shrinking procedure in any order.

Denote by $G_R = (V_R, E_R)$ the graph obtained by applying the heuristic for 2-matching constraints and let $\bar{x}^R$ be the vector of edge weights of G_R.

Lemma 12

(a) *G_R has an even number of odd vertices.*

(b) *If G_R has only even labeled vertices, then the point $\bar{x}$ satisfies all 2-matching constraints.*

(c) *If G_R has an isolated odd labeled vertex, then the vertex set of $G(\bar{x})$ corresponding to that vertex defines a set H and the edges connecting H to $V-H$ define a set T such that (9) holds, i.e. they define a 2-matching constraint which is violated by the point $\bar{x}$.*

(d) *There exists a minimum odd cut-set with capacity less than 1 in $G^*(\bar{x})$ if and only if there exists a minimum odd cut-set with capacity less than 1 in the graph G_R^* obtained by applying the labeling procedure with the graph G_R as input, using, however, in Step 1 only edges $e \in E(\bar{x}) \cap E_R$.*

(e) *The construction of an odd cut-set of capacity less than 1 in $G^*(\bar{x})$ from such a cut-set in G_R^* can be done in a manner analogous to the remarks made after Lemma 11.*

In other words, the 2-matching heuristic does about the same for 2-matching constraints as the shrinking heuristic does for subtour elimination constraints. However, the latter changes the graph in a more fundamental way by adding edges and can thus be expected to bring about a stronger reduction of the size of the graph $G(\bar{x})$.

While the above heuristic has still to be tested in actual computation, the following one has been used with success for the primal facet-identification problem by Padberg & Hong [1980]. It can, however, be adapted easily for the comb problem as well. This second heuristic is essentially enumerative and is derived from the following consideration: Suppose the point $\bar{x}$ for which the comb problem has to be solved is an extreme point of the feasible set of (1) which is *adjacent* to some 0–1 extreme point of the same set. Then

there always exists a 2-matching constraint which is violated by $\bar{x}$ and satisfied by x^* at equality, i.e. there exists a 2-matching constraint which answers the first part of the primal facet identification problem. The reason is that the partial graph $G_F = (V_F, E_F)$ induced by the edges $e \in E$ with $0 < \bar{x}_e < 1$ consists of an even number of disjoint odd cycles (cf. Exercise 8 of Chapter 8). Every odd cycle has an odd number of edges e incident to it which satisfy $\bar{x}_e = 1$. Consequently, choosing H to be the vertex set of an odd cycle in G_F and T to be the collection of edges incident to it and having $\bar{x}_e = 1$, we obtain an even number of 2-matching constraints such that (9) holds and which solve the primal problem.

To generalize this construction, we first determine for a point $\bar{x} \in \mathbb{R}^E$ satisfying (1) the graph $G_F = (V_F, E_F)$. (Note that G_F is different from the graph $G(\bar{x})$ used otherwise.) By tedious but elementary considerations one can show that not only different connected components, but under certain circumstances, also the blocks of G_F play a role in determining a 2-matching constraint. To this end we use an algorithm by Paton [1971] to determine all blocks and cut vertices of G_F. (A *cut vertex* of G_F is a vertex whose removal increases the number of connected components of G_F by at least one.) The resulting blocks with at least three vertices are then used as candidate sets for the set H in the 2-matching constraint. For each set H we identify a set of edges T satisfying (a), (b) and (c) of the 2-matching problem such that their total weight is as large as possible. Then the resulting 2-matching constraint is checked to see if it is satisfied at equality by x^* and cuts off $\bar{x}$. The heuristic attempts to perform the constraint generation starting with the smallest cardinality candidate set H.

Even though this procedure does not guarantee a solution to the primal facet-identification problem, it has been found to be very effective in identifying 2-matching constraints. In judging the computational work involved, it should be kept in mind that the graph G_F is extremely sparse and generally has only a very small number of vertices. Furthermore, lengthy considerations show that one can always cut off points $\bar{x}$ which are such that the face of minimum dimension of the feasible set of (1) containing both x^* and $\bar{x}$ has dimension two.

The second heuristic is best illustrated by an example and we ignore the aspect of the primal facet-identification problem. Consider the graph $G(\bar{x})$ of Figure 9.4. The graph induced by the fractional edges has two connected

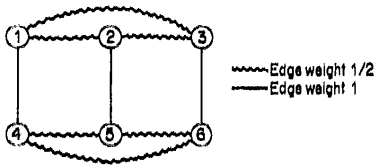

Figure 9.4 Graph illustrating the heuristics

components and the second heuristic finds the 2-matching constraint with $H=\{1,2,3\}$ and $T=\{\{1,4\},\{2,5\},\{3,6\}\}$ which solves the comb problem. Of course, the heuristic given above also finds the same constraint.

If either the first or the second heuristic is applied to the graph of Figure 9.1, no violated 2-matching constraint is found. In contrast to the first 2-matching heuristic, however, the second heuristic does not permit us to conclude that the point $\bar{x}$ depicted in Figure 9.1 satisfies all 2-matching constraints.

3.3 A heuristic for comb constraints

For comb constraints we have at present only a heuristic procedure whose effectiveness remains to be tested in numerical calculations. It will, however, be seen to be useful in the sense that it permits us for instance to identify a comb constraint to cut off the fractional solution of Figure 9.1, thereby giving a list of facet-inducing linear inequalities which permit the solution of this well-known instance of the TSP entirely by the linear programming means described in this chapter.

Like the first 2-matching heuristic, our heuristic procedure for comb constraints is tailored to the TSP, i.e. we assume that the point $\bar{x}\in\mathbb{R}^E$ which we want to cut off satisfies (1) as well as all subtour elimination constraints. As in the case of the 2-matching constraints we do not know how to solve the general case.

One can derive a generalization of (10) for general comb constraints by summing the equations

$$\begin{aligned} &x(\delta(v))=2 && \text{for all } v\in V,\\ &x(E(W))+t_W=|W|-1 && \text{for all } \emptyset\neq W\subset V,\\ &x_e\geqq 0,\ t_W\geqq 0 \end{aligned}$$

over all vertices v in the set H of a comb and adding to it all equations for $W=T_i$, $T_i\cap H$ and T_i-H, respectively, where $i=1,\dots,k$. For odd k this yields

$$2x(E(H))+x(H:V-H)+\sum_{i=1}^{k}(x(E(T_i))+x(E(T_i\cap H))+x(E(T_i-H)))$$
$$+\sum_{i=1}^{k}(t_{T_i}+t_{H\cap T_i}+t_{T_i-H})=2s(C)+1.$$

Consequently, for the comb $C=E(H)\cup\bigcup_{i=1}^{k}E(T_i)$ we have that

$$\bar{x}(C)>s(C) \tag{12}$$

holds if and only if

$$\bar{x}(H:V-H)-\sum_{i=1}^{k}\bar{x}(H\cap T_i:T_i-H)+\sum_{i=1}^{k}(t_{T_i}+t_{H\cap T_i}+t_{T_i-H})<1 \tag{13}$$

holds. The inequality (13) imposes restrictions on the total slack of the subtour elimination constraints, but a graphical interpretation as in the special case where $|T_i|=2$ for $i=1,\dots,k$, does not appear to be easy, especially in view of the exponentially many subtour elimination constraints. Despite that fact, we *conjecture* that there exists a polynomial algorithm for the resolution of the comb problem.

The basic idea of our heuristic for comb constraints is to reduce the problem to the 2-matching problem in a smaller graph obtained by shrinking certain parts of $G(\bar{x})$ into pseudo-vertices using the first shrinking procedure (Section 2).

We have seen that in the subtour elimination case all edges with weight 1 could be shrunk away in any order without loosing the relevant properties of the graph. In the 2-matching case, however, this is not possible. In order to show this, the reader is referred to Exercise 27(b), which gives appropriate examples. But there are other properties that violated combs must satisfy, and these are summarized in the following three lemmas.

Lemma 13 *Suppose that $\bar{x}\in\mathbb{R}^E$ satisfies (1) and that $\bar{x}(E(W))\leqq|W|-1$ for all proper nonempty subsets W of V. Let G_1 be obtained from $G(\bar{x})$ by shrinking all 1-paths in $G(\bar{x})$ (cf. Section 3.2) to 1-paths having exactly one edge. Then there exists a violated comb in $G(\bar{x})$ if and only if there exists a violated comb in G_1.*

Lemma 14 *Suppose that $\bar{x}\in\mathbb{R}^E$ satisfies (1) and that $\bar{x}(E(W))\leqq|W|-1$ for all proper nonempty subsets W of V.*

(a) *If there exists a comb with sets $H, T_1,\dots,T_k\subseteq V$ such that $k\geqq 3$ and (12) hold, then $\bar{x}(E(H))<|H|-1$.*

(b) *If $\bar{x}\in\mathbb{R}^E$ satisfies $\bar{x}(E(W))\leqq|W|-1.5$ for all proper nonempty subsets $W\subseteq V$, then there exists no comb C such that (12) holds.*

(c) *If (12) holds for some comb C with sets $H, T_1,\dots,T_k\subseteq V$, then $\bar{x}(T_i\cap H:T_i-H)>0$ and $\bar{x}(E(T_i))>|T_i|-2$ for $i=1,\dots,k$.*

(d) *If $e=\{u,v\}$ is such that $\bar{x}_e=1$ and there exists a comb which is violated by $\bar{x}$, then there exists a violated comb with sets $H, T_1,\dots,T_k\subseteq V$ such that either $\{u,v\}\in H-\bigcup_{i=1}^{k}T_i$ or $\{u,v\}\in T_i$ for exactly one $i\in\{1,\dots,k\}$.*

Lemma 15 *Suppose that $\bar{x}\in\mathbb{R}^E$ satisfies (1) and that $\bar{x}(E(W))\leqq|W|-1$ for all proper nonempty subsets W of V. Let G_1 be obtained from $G(\bar{x})$ by shrinking all 1-paths in $G(\bar{x})$ to 1-paths having exactly one edge. Let $W\subseteq V$ be such that $\bar{x}(E(W))=|W|-1$ and either $|V|\geqq 7$, $|W|=3$ and $\bar{x}_{uv}=1$ for $u,v\in W$, or $|V|\geqq 8$, $|W|=4$ and $\bar{x}_{uv}=\bar{x}_{rs}=1$ where $W=\{r,s,u,v\}$. Let G^* denote the graph obtained from G_1 by shrinking vertices u and v to a single pseudo-vertex using the shrinking procedure from Section 2, and in the case where $|W|=4$, by shrinking vertices r and s to a single pseudo-vertex using this procedure as well. Then there exists a violated comb in G_1 if there exists a violated comb in G^*.*

The preceding considerations give rise to the following heuristic for the identification of comb constraints. We will assume that the point $x \in \mathbb{R}^E$ satisfies all subtour elimination and 2-matching constraints.

A heuristic for comb constraints

Input: The graph $G(\bar{x})$ as defined in Section 2.1.

Step 1. Shrink all 1-paths in $G(\bar{x})$ as in Lemma 13 and replace $G(\bar{x})$ by the shrunk graph.

Step 2. Find a subset $W \subseteq V$ with $|W| \leq \lceil n/2 \rceil$ such that $\bar{x}(E(W)) = |W| - 1$ holds (choose $|W|$ as large as possible). If no such set W exists, stop.

Step 3. Find a proper subset $U \subseteq W$ such that $\bar{x}(E(U)) = |U| - 1$ holds (again choose $|U|$ as large as possible). If no such U exists, set $U = \emptyset$.

Step 4. Shrink all edges in $E(U)$ and all edges in $E(W-U)$, respectively, to single pseudo-vertices using the shrinking procedure from Section 2 and solve the 2-matching problem in the reduced graph.

Step 4.1. If a violated 2-matching constraint is found, expand it to the original graph $G(\bar{x})$ and stop; the constraint thus found solves the comb problem.

Step 4.2. If no violated 2-matching constraint exists go to Step 5.

Step 5. Replace $G(\bar{x})$ by the shrunk graph. If the graph has fewer than seven vertices, stop. Else go to Step 1.

To illustrate the heuristic, consider the graph in Figure 9.1. Carrying out Step 1, we obtain the graph of Figure 9.5.

Now $n = 15$ and the largest subset of vertices given by Step 2 is $W = \{10, 11, 12, 24, 25, 26, 27, 28\}$, and Step 3 yields $U = \{10, 24, 25, 26, 27, 28\}$. We shrink $G(\bar{x})$ using the shrinking procedure and obtain the graph of Figure 9.6.

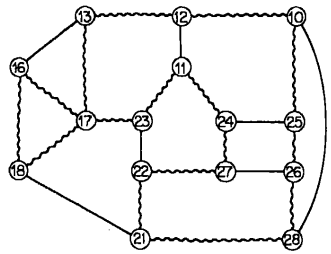

Figure 9.5 Shrunk graph from Figure 9.1

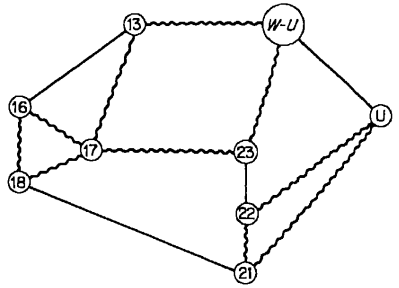

Figure 9.6 Shrunk graph from Figure 9.5

Now applying the heuristic for 2-matching constraints, we find that the comb with

$$H = \{1, 2, \ldots, 10, 21, 22, 24, 25, \ldots, 42\},$$
$$T_1 = \{20, 21\},$$
$$T_2 = \{22, 23\},$$
$$T_3 = \{1, 2, \ldots, 12, 24, 25, \ldots, 42\}$$

is violated by the point $\bar{x} \in \mathbb{R}^{862}$ depicted in Figure 9.1. (Note that this comb constraint was expanded from the 2-matching constraint determined by $\bar{H} = \{U, 21, 22\}$, $\bar{T}_1 = \{18, 21\}$, $\bar{T}_2 = \{22, 23\}$, $\bar{T}_3 = \{W - U, U\}$.) Of course, the comb with

$$H' = \{11, \ldots, 20, 23\}$$

and $T_1' = T_1$, $T_2' = T_2$, $T_3' = T_3$ is equivalent to the former, requires less storage space and thus should be stored in actual numerical calculations.

We have recomputed the 42-city problem of Dantzig, Fulkerson & Johnson [1954] using a code currently under development in Augsburg, in which the solution methods for the identification of subtour elimination constraints described in Section 2.2 and for the identification of 2-matching constraints described in Section 3.1 are implemented. It turns out that the solution found by Dantzig *et al.* shown in Figure 9.1 is an optimum solution (of length 698) over the polytope given by the degree equation, the upper and lower bounds on the edges, the subtour elimination constraints and the 2-matching constraints. If the comb inequality found by the heuristic described above is added, the resulting linear system has a tour (of length 699) as an optimum solution. Thus the 42-city problem can be solved with the cutting plane generation techniques known at present without recourse to branch and bound.

We leave it as an exercise for the reader to prove that any violated 2-matching constraint obtained in Step 4.1 of the above heuristic yields a

violated comb inequality when expanded to the original graph $G(\bar{x})$. The idea of the heuristic reverses in an interesting way the proof ideas of the various lifting theorems proved by Grötschel & Padberg [1979b]. While we have not yet been able to come up with a heuristic for combs which shares the properties of the subtour and 2-matching heuristics, it appears that nested families of subtour elimination constraints which are satisfied by the point $\bar{x} \in \mathbb{R}^E$ play a role in solving the comb problem and that, possibly, the ideas underlying the heuristic can be generalized to solve the comb problem. On the other hand, Exercise 30(b) shows that it is a *heuristic* procedure; that is, it is not guaranteed to solve the comb identification problem.

Exercises

17. Prove Lemma 7.
18. Apply the labeled graph construction to the graph of Figure 9.1.
19. Prove Lemma 8.
20. Prove Lemma 9.
21. Prove Lemma 10.
22. Prove Lemma 11.
23. Prove Lemma 12.
24. Apply the 2-matching heuristic to the graph of Figure 9.1.
25. As was mentioned in this section, although we have a polynomial algorithm to solve the 2-matching problem under the condition that $\bar{x} \in \mathbb{R}^E$ satisfies (1), no such algorithm is known without this additional assumption. Thus it would be of interest to consider the following *open problems.*
(a) What are the extreme points of the linear system

$$x(E(H)) + x(T) \leq |H| + \lfloor \tfrac{1}{2}|T| \rfloor \qquad \text{for all } T \in \mathcal{T}_H \text{ and } H \in \mathcal{G},$$
$$x_e \geq 0 \qquad \text{for all } e \in E,$$

where $\mathcal{G}$ is a family of subsets $H \subseteq V$ with the property that $H \in \mathcal{G}$ if and only if $V - H \notin \mathcal{G}$ (or $\mathcal{G} = 2^E$) and for $H \in \mathcal{G}$ the family $\mathcal{T}_H$ is the collection of sets T of edges of E satisfying (a), (b) and (c) of the 2-matching problem?
(b) Solve the 2-matching problem for an arbitrary point $\bar{x} \in \mathbb{R}^E$, i.e. the problem of testing feasibility or finding a violated inequality for the inequality system of part (a).
26. Analogously to Exercise 25, we formulate the following *research problem.*
(a) What are the extreme points of the linear system

$$x(C) \leq s(C) \qquad \text{for all combs } C \in \mathcal{C},$$
$$0 \leq x_e \leq 1 \qquad \text{for all } e \in E,$$

where $\mathcal{C}$ is the family of all combs in the undirected complete graph?
(b) Solve the comb problem for an arbitrary point $\bar{x} \in \mathbb{R}^E$, i.e. the problem of testing feasibility or finding a violated inequality for the inequality system of part (a).

(c) Solve the comb problem for a point $\bar{x} \in \mathbb{R}^E$ satisfying $\bar{x} \in Q^n_{1T} \cap Q^n_{2M}$.
27. (a) Prove Lemma 13.
(b) Using the graph $G(\bar{x})$ of Figure 9.3, show that if the edges $\{1, 2\}$ and $\{5, 6\}$ are shrunk using the shrinking procedure of Section 2, then the resulting weight vector violates a 2-matching constraint. However, if the edges $\{1, 2\}$ and $\{3, 4\}$ are shrunk using this procedure, then the resulting weight vector is a convex combination of incidence vectors of 2-matchings (in the smaller graph).
28. Prove Lemma 14.
29. Prove Lemma 15.
30. (a) Apply the comb heuristic to the graph of Figure 9.3.
(b) Apply the comb heuristic to the graph of Figure 8.5(a) of Chapter 8.
31. (*Research problem*) Consider the following possible extension of Lemma 14(d). Prove or disprove: If $W \subseteq V$ satisfies $\bar{x}(E(W)) = |W| - 1$ and $\bar{x}(E(W^*)) < |W^*| - 1$ for all proper subsets W^* of W with $|W^*| \geq 2$ and if there exists a comb which is violated by $\bar{x}$, then there exists a violated comb with sets $H, T_1, \ldots, T_k \subseteq V$ such that either $W \subseteq H - \bigcup_{i=1}^k T_i$ or $W \subseteq T_i$ for exactly one $i \in \{1, \ldots, k\}$.
32. (*Research problem*) Suppose $\emptyset \neq W \subseteq V$ and $\emptyset \neq U \subseteq V$ satisfy $U \cap W = \emptyset$ and both U and W satisfy the assumptions of Exercise 31 above. Furthermore, suppose that $\bar{x}(E(U \cup W)) = |U \cup W| - 1$ holds. Let G^* denote the graph obtained from G_1 by shrinking all edges in $E(U)$ and in $E(W)$, and assume that G^* has at least six vertices. Find conditions on U and W under which the following statement is true: there exists a violated comb in G_1 if and only if there exists a violated comb in G^*.

4 COMPUTATIONAL EXPERIMENTS

This is the *show and tell* section of our two-part survey of polyhedral aspects of TSPs. The objective of *optimization* is to solve actual problem instances and thus the ultimate test of a mathematical theory for such problems is whether or not it aides the numerical aspect of problem solving. While the story is by no means told, we give here an account of what has been done to date in terms of applying the polyhedral theory for the TSP to numerical problem solving. This includes the solution to optimality of the largest *symmetric* TSP known to have been solved, but does not include any experiments on the asymmetric TSP, as we are not aware of any computational study that uses the theoretical results of Section 5 of Chapter 8 in a consistent way.

As a result, we restrict our presentation to a brief summary of four papers [Crowder & Padberg, 1980; Grötschel, 1977a, 1980b; Padberg & Hong, 1980]. There are, however, several other studies under way which utilize polyhedral information in the actual solution of symmetric TSPs such as the solution of a 125-city problem done by students of Professor W. R. Pulleyblank at the University of Grenoble (France) and the solution of a

68-city problem done by students of Professor L. Wolsey at the Catholic University of Louvain-la-Neuve (Belgium).

Related studies concerning the use of cutting planes in the solution of symmetric TSPs have been carried out recently [Land, 1979; Miliotis, 1976, 1978; Fleischmann, 1981, 1982], and earlier [Hong, 1972]. While these studies reported good results for medium sized TSPs, their methodology does not focus on testing the polyhedral theory developed here, which is the intent of our brief survey.

For an application of polyhedral theory to the solution of large scale 0–1 programming problems having no special structure such as implied by the TSP, the reader is referred to the paper by Crowder, Johnson & Padberg [1983], where computation has confirmed the hypothesis that facet-inducing linear inequalities are an indispensable tool in the numerical solution of hard combinatorial optimization problems in addition to the TSP; see also Grötschel, Jünger & Reinelt [1984] and Barahona & Maccioni [1982].

4.1 Solution of a 120-city problem

The first instance of a symmetric TSP that was solved using only (a subset of the) clique tree inequalities as defined in Section 4.2 of Chapter 8 (namely subtour elimination, 2-matching and comb constraints) is a 120-city problem given by the data in the *Deutscher Generalatlas* (Mairs Geographischer Verlag, Stuttgart, 1967–68). Grötschel [1977a, 1980b] did not need to resort to branch and bound or other enumerative methods in order to obtain an optimal tour, i.e. the relaxation method for combinatorial optimization problems was used and nothing else.

To implement the procedure, the facet identification step (Step 3) of the procedure was carried out *visually*, i.e. the optimal solutions x^k of the linear programming relaxation obtained in Step 2 were plotted and suitable violated constraints from among those which define facets of Q_T^{120} were chosen by inspection. Thus this implementation is very much in the spirit of the work in Dantzig, Fulkerson & Johnson [1954], except that a better understanding of the underlying theory of the polyhedral aspects of TSPs was available, i.e. more facet-inducing linear inequalities were known, and better computers and better linear programming software could be used. Furthermore, rather than using 'pegs and strings' in order to obtain a heuristic solution to the problem as was done by Dantzig, Fulkerson & Johnson [1954], computer programs were used to find a round trip of length 7011 km which was rather close to the optimal tour of length 6942 km for this particular problem.

In order to find and to prove the optimality of the tour shown in Figure 9.7, thirteen iterations of the relaxation method were necessary. In this process only 96 subtour elimination and comb constraints of the total universe of 10^{179} such constraints were generated to establish optimality of

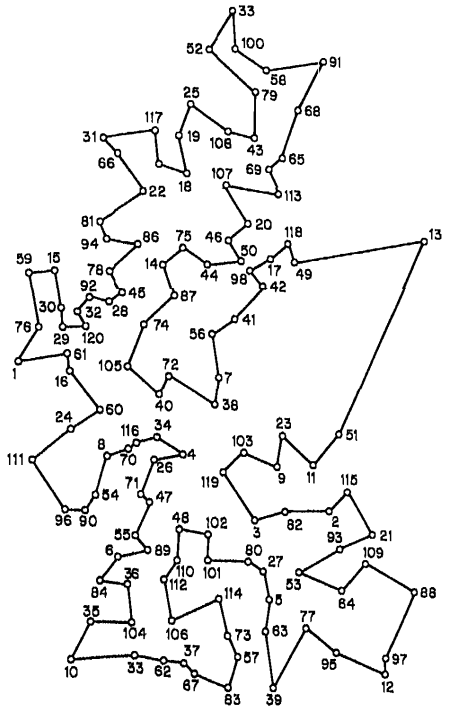

Figure 9.7 Solution of a 120-city problem

the tour (cf. Exercise 2). More precisely,

36 subtour elimination constraints,
25 2-matching constraints and
35 (other) comb constraints

were needed to find and to prove optimality of the tour displayed in Figure 9.7. Table 9.2 gives the results of the thirteen iterations.

In Table 9.2, z_{LP} is the optimal objective function value of (LP_k) and #c is the number of constraints identified visually to chop off the optimal solution x^k of (LP_k) in the relaxation method for combinatorial optimization problems when applied to this particular problem, where $k = 1, \ldots, 13$.

The CPU times needed for the solution of the thirteen linear programs

Table 9.2 120-city problem

Run no.	z_{LP}	#c
1	6,662.5	13
2	6,883.5	15
3	6,912.5	7
4	6,918.75	9
5	6,928	6
6	6,935.3	9
7	6,937.222	8
8	6,939.5	5
9	6,940.383	4
10	6,940.816	12
11	6,941.184	5
12	6,941.5	3
13	6,942	0

using IBM's MPSX/370 on the IBM computer 370/168 of the Rechenzentrum der Universität Bonn ranged between 30 seconds and 2 minutes, the number of pivot operations was between 100 and 1,000, and both CPU times and pivot operations increased slightly but not monotonically with the number of additional inequalities. The last run, for instance, was executed in 1.76 CPU minutes and 714 pivot operations were necessary to obtain the optimal solution.

Table 9.2 shows that the objective function values z_{LP} of the linear programs increase monotonically. Moreover, the gains in the objective function value are much larger in the beginning than in the later iterations. This is a behavior one should expect based on the existing computational experience with general cutting plane procedures (Gomory cuts, etc.). In these methods usually a 'tailing off' can be observed where no or almost no increase in the objective function value occurs during quite a number of iterations. This is due partly to the possibility of alternate optima of the relaxed problems which the Gomory (or other) cuts do not take care of. This behavior has been observed much less frequently when facet-incuding inequalities are added, as in the TSP example we discuss.

There is a further – one may say philosophical or tactical – difference between the two approaches. A traditional cutting plane procedure is a mechanism which creates a cutting plane that cuts off x^k whenever x^k is fractional. The cutting planes obtained this way are not taken from a class of inequalities defined beforehand and need not even be supporting hyperplanes of the underlying integer polytope (see Padberg [1973] for a relevant example), though for those cutting plane generation schemes for which finiteness has been established all facets will be generated after a finite number of iterations. The (empirically observed) trouble with these procedures is that often the cutting planes generated in the beginning are quite

weak and that it takes some time to get a good approximation of the underlying integral polytope. Thus the empirical convergence is rather slow.

In contrast to this, in the approach described here, we use only cutting planes from well-defined classes of facet-defining inequalities. The approach is very problem-specific and uses the structural properties of the underlying combinatorial optimization problem. For points x^k not in Q_T^n we only generate cutting planes from the classes of inequalities we intend to consider (subtour elimination constraints, 2-matching constraints, etc.). We know from our theoretical investigations that these are in a precise sense best possible cutting planes and that they are not 'far away' from Q_T^n (which could be the case with Gomory-cuts). Moreover, we do not continue forever which, in principle, might be possible with 'general cuts'. We only check the facet-defining inequalities in specific classes and if a cutting plane for x^k from these classes ceases to exist we give up and default to branch and bound. Our computational experience has shown that not too many iterations are needed to reach this point and that the final branch and bound step can be finished very quickly. Moreover, we also recommend that branch and bound should only be used after the classes of facet-defining cutting planes are exhausted. This recommendation may be a surprise but its usefulness is also backed by other computational experiments like the ones reported by Crowder, Johnson & Padberg [1983] and Grötschel, Jünger & Reinelt [1985].

4.2 Computation of lower bounds for 74 TSPs

This section is based on the work of Padberg & Hong [1980] where a sample of 74 symmetric TSPs was used to assess the computational value of facet-inducing linear inequalities for the solution of TSPs. We note that out of the total 74 problems, only 54 were solved (to optimality), but in all cases excellent lower bounds were obtained.

To compute the lower bounds a *primal* cutting-plane procedure was implemented rather than the relaxation method. As explained in Section 1.4, subsequent computational experience has convinced us that there is no reason to prefer a primal approach. Indeed, it would have been simpler to implement the relaxation method, since this would have avoided the necessity of writing a simplex code of its own.

All problems were first run with the heuristic of Lin & Kernighan [1973]. As it turned out, this carefully designed heuristic found excellent, often optimal solutions.

The core of the linear programming procedure is the primal simplex algorithm with bounded variables. The program is started by reading in the distance table of the problem, or the coordinates of the n points in the case of Euclidean TSPs and, in most cases, the heuristic solution obtained earlier. This starting solution is then used to initialize the linear programming problem

$$\min\{c^T x \mid Ax = \mathbf{2},\ 0 \leqq x \leqq \mathbf{1}\} \tag{14}$$

where c^T is the vector with $m = n(n-1)/2$ components given by the edge distances, A is the $n \times m$ incidence matrix of the complete graph having n vertices, $\mathbf{2}$ is the vector with n components equal to 2, and $\mathbf{1}$ is the vector with m components equal to 1. The program then proceeds by adjoining cuts (i.e. additional constraints) in the *primal* fashion: If the next feasible solution to the current linear program is the incidence vector of a tour, a usual pivot is carried out. (This includes the case of degeneracy, where the *next* feasible solution is identical to the current one.) Otherwise, the next feasible solution must be chopped off by some cutting plane which is satisfied by the current solution at equality. Subroutines are called which attempt to identify suitable subtour elimination, 2-matching and comb constraints. If suitable constraints are identified, the current linear program is enlarged by adjoining those cuts and a pivot is executed on the enlarged problem. At this point one has a *tighter* linear programming relaxation of the TSP than the one given by (14), since *all* tours satisfy the new constraints but a portion of the feasible space of (14) is chopped off. One can iterate this procedure and since there are only finitely many subtour elimination and comb constraints, termination is guaranteed. If the program halts by displaying optimality of the current linear programming problem, the optimality of an incidence vector of a *tour* has been proved, and the TSP in question has been solved. (This was the case in 54 of 74 sample problems.) Otherwise, the program, which is called *TSP code*, eventually encounters the situation where a suitable constraint cannot be found. One has no choice but to default, i.e. to solve the current *linear* program to optimality and thereby get a lower bound on the minimum tour length. (The alternative to use branch and bound is described in the next section.)

The overall objective of the computational study was to validate empirically the usefulness of facet-inducing inequalities in the solution of TSPs. In order to evaluate the value of facet-inducing inequalities towards the goal of proving optimality, one proceeds as follows: Given a heuristically obtained solution, the 2-matching problem is solved as a *linear* program, i.e. the constraints are given by the system (1) (without the integrality stipulation). Using the same solution as a starting solution, the problem is run a second time generating facet-inducing inequalities. The second run either terminates with an optimal tour or, in case a suitable new constraint is not found, defaults to solving the by now enlarged linear programming problem. This gives two values: VALUE1 is the objective function value without cuts (problem (14)) and VALUE2 is the objective function value with cuts. If TOUR denotes the minimum length tour of the problem, then the following ratio is a good proxy for measuring the added value of the additional work:

$$\text{RATIO} = (\text{VALUE2} - \text{VALUE1})/(\text{TOUR} - \text{VALUE1}).$$

The value of RATIO can vary between 0 and 1, where 0 indicates that the additional cuts did not strengthen the lower bound, and 1 indicates that the additional cuts were sufficient to solve the problem to optimality. The measure RATIO is invariant under any scaling and translating of the data. It is a *conservative* measure, since only an upper bound on TOUR maybe known. In this case, the value of the additional cuts is not overestimated. Since the choice of the *starting solution* is known to greatly affect the performance of simplex methods, the same starting solution was used in the computation of both VALUE1 and VALUE2. Due to this choice it is to some extent meaningful to evaluate the trade-off between added value and additional work also in terms of increased CPU times and increased number of pivots. Finally, all CPU times reported include the entire set-up of the problem, and in particular, a very costly ordering of all edges of the complete graph with respect to the reduced cost of an initial basis which is set up with reference to the starting solution. (This feature of the program was dropped for the largest sample problem having 318 cities.)

55 Randomly generated Euclidean TSPs

To generate the problems for this part of the computational study the pseudo-random number generator of Lin & Kernighan [1973] was used which generates coordinates x_i and y_i with values between 1 and 1,000 for $i = 1, \ldots, n$. Due to the fact that coordinates are generated as pairs, the same random seed for $n+m$ cities produces a graph that is properly contained in the graph on the first n cities, thus making it possible to study how increasing n affects the added value of the facet-inducing inequalities. Ten different problems were run for $n = 15$, 30, 45, 60 and 75, respectively using ten different seeds for the random number generator. Furthermore, five Euclidean problems with $n = 100$ due to Krolak, Felts & Marble [1971] were included in this statistical part of the study, though they are different from the other ones.

Table 9.3 contains all the relevant statistics for this part of the study. The entries in Table 9.3 were obtained by averaging the respective individual figures and their mean μ is given with the standard deviation σ. The top row of Table 9.3 contains the value RATIO. As it is to be expected, RATIO declines with increasing n. TOUR is the tour length obtained by the heuristic.

The bottom line of Table 9.3 (OPTIM) specifies the number of times the linear program terminated by proving optimality of the heuristically obtained tour. GAP1 measures the average difference between TOUR and the objective function value of the (initial) linear program (14). GAP2 is the crucial measure in evaluating the constraint-generation procedure. It is the difference between TOUR and the objective function value of the amended linear program. For example, when $n = 100$ GAP2 averaged 120, and thus an average lower bound of 21,387 was shown for tours with mean length 21,507. Note that RATIO is then just (GAP1 − GAP2)/GAP1. PIVOT2 is the average of the total number of pivot operations carried out by the

Table 9.3 55 Euclidean TSPs

n		15	30	45	60	75	100
RATIO	μ	1.0	0.99	0.93	0.92	0.88	0.92
	σ	0.0	0.03	0.11	0.10	0.09	0.02
TOUR	μ	3,555	4,738	5,566	6,297	6,878	21,507
	σ	383	314	224	181	224	525
GAP1	μ	224	352	379	452	387	1507
	σ	121	100	149	133	93	313
GAP2	μ	0.0	5	24	38	50	120
	σ	0.0	15	57	44	44	43
PIVOT2	μ	11	34	47	76	87	167
	σ	2	7	13	30	23	40
Δ PIVOT	μ	2	13	17	36	37	97
	σ	2	5	12	29	22	38
TIME2*	μ	0.33	1.37	4.46	14.47	30.52	108.74
	σ	0.03	0.26	1.27	6.82	10.81	39.97
Δ TIME*	μ	0.07	0.46	1.39	6.25	11.64	50.4
	σ	0.03	0.23	1.23	6.63	10.63	31.7
CUTS	μ	3	12	15	26	28	72
	σ	2	5	8	13	12	18
OPTIM		10	9	5	4	3	0

* Seconds, IBM 370/168 MVS

constraint-generating program. Δ PIVOT is the average increment of the pivot count over what it takes to solve the initial linear program. Likewise, TIME2 specifies the total CPU time of the constraint-generating program, and Δ TIME the average increment over the respective times for the initial linear program. (The numbers in the rows labeled σ are the respective standard deviations for the sample problems.) Finally, CUTS is the average number of constraints that were generated and amended to the original linear program. Thus for 100-city problems the initial linear program has 100 rows and 4,950 variables, while at termination of the constraint-generation procedure the linear program increased on average to 172 rows and 5,022 variables, a truly modest increase given the complexity of the problem and the tightness of the bound obtained. For the individual figures for all of the sample problems, the reader is referred to the original paper [Padberg & Hong, 1980].

Traps for the traveling salesman

The heading for this section is taken from a talk by Papadimitriou & Steiglitz [1967]. Papadimitriou and Steiglitz demonstrated the difficulty that local search procedures for combinatorial problems (more precisely, exchange heuristics) can encounter [Papaditimitriou & Steiglitz, 1977]. They construct a class of non-Euclidean TSPs for which local search procedures

find the optimum tour only if computational work tantamount to total enumeration is carried out. In fact, by choosing a cost parameter appropriately, the best solution found by a local search heuristic can be made 'arbitrarily bad'. More precisely, the edge weights of these graphs are 0, 1, M or $2M$, respectively, and there are

$$\frac{1}{32}n^2+\frac{3}{4}n+2$$

edges having weights 0 and 1. These test problems were run both with the program made available by Shen Lin and with the TSP code. The TSP code was run with its initial tour chosen both arbitrarily and as the result of the heuristic, as was done above. To get a representative picture, 11 problems ranging from $n=40$ to $n=120$ were executed. In all but one of the 22 runs an optimal tour was found. (The exception occurred for $n=104$ and a randomly selected starting tour.) The complete details of this work are given by Padberg & Hong [1980].

Eight test problems from the literature

In order to permit a limited comparison of the performance of the constraint-generation procedure *vis-à-vis* other approaches a number of test problems that have been used by other researchers were solved. The results are summarized in Table 9.4.

The largest problem, LIN318, is a 318-city problem, and the data were published by Lin & Kernighan [1973]. The data come from an actual problem involving the routing of a numerically controlled drilling machine through three identical sets of 105 points each plus three outliers. As the drilling is done by a pulsed laser, drilling time is negligible and the problem

Table 9.4 Eight problems from the literature

	DAN42	GRO48	HEL48	TOM57	KROL70	GRO120	KNU121	LIN318
RATIO	1.0	0.95	1.0	0.95	0.98	0.99	0.76	0.96
TOUR	699	5046	11461	12955	675	6942	349	41349
GAP1	58	277	264	321-1/2	51-1/2	279-1/2	21	2583-1/2
GAP2	0	14-15/16	0	15	1-3/4	3-19/26	5-1/2	112-128/240
PIVOT2	37	83	38	61	120	243	74	578
Δ PIVOT	7	50	5	17	67	174	29	327
TIME2*	3.10	9.16	4.30	10.40	31.91	221.50	7.25	1751.46
Δ TIME	0.53	5.07	0.61	2.69	15.58	110.30	2.71	1080.66
CUTS	9	32	10	22	44	75	10	171

* Seconds, IBM-370/168 MVS

REFERENCE: [Dantzig, Fulkerson & Johnson, 1964] | [Grötschel, 1977a (Shell's Residuites of Germany)] | [Held & Karp, 1970] | [Karg & Thompson, 1964] | [Krolak, Felts & Marble, 1971] | [Grötschel, 1977a] | [Knuth, 1976] | [Lin & Kernighan, 1973]

becomes a standard TSP. The only exception from the standard form is that particular start and end points are to be used; the resulting Hamiltonian path problem can, however, easily be accommodated within the linear programming framework by assigning a negative distance to the particular arc. The distance table of the complete graph on 318 points (with the exception of one edge) was computed from the coordinates published by Lin & Kernighan [1973]. The coordinates are given in milli-inches and the usual single-precision square root function was used to calculate the distance. More precisely, one computes the distances by taking the square root of the sum of the squared differences, adding 0.5 to the resulting real number and by subsequently truncating it to its integer part. While the resulting total path length differed by approximately 10 milli-inches from the total path length that results from adding up the 317 individual real numbers, it was felt that this difference was small enough to justify the approximation. If the best solution published earlier is calculated this way, it is 41,871 milli-inches (rather than 41,883 milli-inches) and the best solution that was found after several runs by a changed TSP code has 41,349 milli-inches. It was thus 522 milli-inches shorter than the solution given in Lin & Kernighan [1973] and as Table 9.4 shows, it is at most $\frac{1}{4}\%$ off the shortest possible Hamiltonian path through the 318 points with distances as defined above.

The changes in the TSP code included a form of *sequential optimization* in order to accommodate the 50,403 variables over which the optimization has to be carried out. That is, a sparse subgraph of only *short* edges was explicitly considered in the optimization and upon termination, the *long* arcs were checked by a subroutine to see if they price out correctly. If not, then (automatically) a new round was initiated; that is, the constraint-generation program was called again.

The original paper [Padberg & Hong, 1980] contains additional details on the implementation and actual computer runs that we cannot reproduce here because of space limitations.

4.3 Solution of ten large-scale TSPs

The most surprising outcome of the computational study described in Section 4.2 is that only *very few additional facet-inducing inequalities* are needed in order to obtain excellent lower bounds on the minimum tour length, and in some cases, to prove optimality as well. The results obtained in the statistical part of the computational study are consistently at most $\frac{1}{2}\%$ off the optimal tour length and the standard deviations are consistently small as well. The test problems which proved insoluble for the local search procedure were solved to optimality without any difficulties. The results for the test problems from the literature including the – by today's standards – truly large-scale TSPs with 120 and 318 cities, respectively, generally outperformed the results that one might expect based on the statistical part of the study. In particular, the bound for the 120-city problem obtained this

way indicates that the solution is within 0.04% of the optimum tour and the bound for the 318-city problem indicates that the solution is within 0.26% of the minimum length Hamiltonian path through the 318 points. With the resulting (remaining) gap between the best tour found and the bound obtained by the use of facet-incuding inequalities being so relatively small, it is entirely realistic to expect that any good branch and bound procedure will enable one to solve large-scale TSPs to optimality. This is what we will describe in this last section of our ongoing saga of the traveling salesman. The material of this section is based on the paper of Crowder & Padberg [1980].

Before using the output of the TSP code described in the previous section as input for a more intricate software system, several changes were carried out to improve the bounds of ten of the large-scale TSPs of the previous section. These changes concerned the identification of additional subtour elimination constraints and the determination of the *hard core* of the respective TSPs that remained to be optimized.

To this end a subroutine was written to fix nonbasic variables of the last LP optimum at either 0 or 1, using the available upper and lower bounds on the optimal objective function value. The basic idea for this *variable fixing* dates back to Dantzig, Fulkerson & Johnson [1954] and is described in detail in the original paper. This procedure is very effective; on average, the number of variables was reduced from $\binom{n}{2}$ by a *factor of 20.*

The ten problems selected from the previous study are shown in Table 9.5 with KRO124 through KRO128 being the five 100-city problems from Krolak, Felts & Marble [1971]. In this table, n is the number of cities and $m = n(n-1)/2$ is the number of variables (=number of edges of the graph). The only exception is KNU121 which is a supersparse problem having 121

Table 9.5 Ten large-scale TSPs

Problem	n	m	n_R	m_R	TIME*	RATIO	LPVALUE
GRO48	48	1,128	86	104	9	0.95	5,031.06
TOM57	57	1,596	89	91	7	0.98	12,948.5
KRO124	100	4,950	187	248	66	0.97	21,225.31
KRO125	100	4,950	170	446	122	0.91	21,978.00
KRO126	100	4,950	183	185	52	0.98	20,730.08
KRO127	100	4,950	192	220	114	0.97	21,257.48
KRO128	100	4,950	203	334	213	0.93	21,970.83
GRO120	120	7,140	199	239	149	0.97	6,934.89
KNU121	121	222	139	182	9	0.90	346.5
LIN318A	318	50,403	496	1,372	2,231	0.97	41,269.83
LIN318B	318	50,403	495	1,144	2,283	0.97	41,269.00
LIN318C	318	50,403	495	1,208	2,295	0.97	412,820.0

* Seconds, IBM 370/168 MVS

cities and only 222 edges. n_R is the number of rows after running the problem with the changed TSP code using the best available tour as a starting solution for the linear program, and m_R is the number of variables that were either basic at the final linear programming optimum or that could *not* be fixed at their respective non-basic value. TIME is the total execution time to obtain LPVALUE at termination. RATIO is the ratio described in Section 4.2. Three runs were made with LIN318, labeled A, B and C. The first was started with the suboptimal solution of length 41,349 obtained earlier, and the second one was started with the optimal tour of length 41,345 obtained after executing LIN318A with the software system described below. The third one has an additional factor of 10 in the data in order to increase the precision of the distance calculations.

Table 9.6 gives a breakdown of the facet-inducing linear inequalities that were generated in the respective runs with the (changed) TSP code to get the linear programs of Table 9.5. The fact that only few (general) comb inequalities were found by the program reflects our limited knowledge of this type of constraint at that time. (The heuristic for comb constraints presented in Section 3 is new and should produce better results.)

After the TSP code is run, the LP obtained has an optimal solution with fractional values. The final phase of the computation consists of several iterations. In each iteration, a branch and bound routine is used to find an optimal 0–1 solution to the current LP. If the 0–1 solution thus found is a tour, the calculation is stopped. Otherwise, subtour elimination constraints are identified by the program to cut off this 0–1 solution, and the next iteration is begun.

The results of this procedure are displayed in Table 9.7. The second and third columns display the problem size: n_R is the number of rows and m_R is

Table 9.6 Breakdown of constraints by type

Problem	Subtour	2-Matching	Comb	Total
GRO48	21	16	1	38
TOM57	16	16	0	32
KRO124	54	32	1	87
KRO125	40	28	2	70
KRO126	43	37	3	83
KRO127	55	36	1	92
KRO128	45	57	1	103
GRO120	51	28	0	79
KNU121	18	0	0	18
LIN318	157	20	0	177
Total	500	270	9	779
Percent	64.3	34.6	1.1	100

Table 9.7 Optimizing ten large-scale TSPs

Problem	n_R	m_R	MIP	NODES	TIME	Δn	NODES	TIME	Δn	NODES	TIME	TOTAL TIME*	OPT TOUR LENGTH
GRO48	86	104	1	17	18	2	3					5	5,046
TOM57	89	91	1	2	3	0	1					6	12,955
KRO124	187	248	1	8	15	2	7					13	21,282
KRO125	170	445	2	133	147	59	70	1	115	146	51	79	159	22,141
KRO126	183	185	1	5	5	1	2						9	20,749
KRO127	192	220	2	4	12	2	7	1	2	12	0	7	21	21,294
KRO128	203	334	1	6	22	5	11						24	22,068
GTO120	199	239	2	17	19	5	8	1	18	20	5	8	24	6,942
KNU121	139	182	3	14	25	1	2	1	15	23	1	2	3	15	23	1	2	16	349
LIN318A	496	1,372	3	22	22	50	52	5	35	36	81	86	1	21	71	75	163	319	41,345
LIN318B	495	1,144	2	41	41	123	125	4	54	58	129	146						321	41,345
LIN318C	495	1,208	2	188	188	349	359	4	150	186	368	445						818	413,589

* Seconds, IBM 370/168 MVS

the number of columns of the (pure) 0–1 problem to be optimized. The column labeled MIP gives the number of overall iterations necessary to come to a halt. The first of the two columns labeled NODES gives the number of nodes of the branch and bound tree generated to find the optimum 0–1 solution, and the second such column gives the total number of nodes generated to prove optimality.

Likewise, the first of the two columns labeled TIME gives the time spent in the branch and bound procedure to find the optimum 0–1 solution, whereas the second one gives the total time spent to prove optimality. (Thus in problem GRO48 it took 17 nodes and 2 seconds of CPU time to find the optimum and one additional node and an additional second to prove optimality.) The times have been rounded to the nearest second. In the column labeled TOTAL TIME is the CPU time required for all aspects of the procedure, both computational and bookkeeping. The last column of Table 9.7 gives the respective optimum tour lengths. If more than one iteration was required, the figure in the column labeled Δn indicates the number of new subtour elimination constraints generated. (Thus in the run LIN318A, after the first iteration, five subtour elimination constraints were violated by the 0–1 solution found and appended to the linear program which now is of size $501 \times 1{,}372$.) Using the previous optimal basis, the enlarged program is reoptimized.

The runs described in this section and the previous one were executed on the IBM 370/168, MVS/TSO at the IBM T. J. Watson Research Center in Yorktown Heights, N.Y. The software packages MPSX/370 and MIP/370 of the IBM Corporation were used for the linear programming and the branch and bound calculations, respectively, for the runs reported in this section.

A first glance at Table 9.7 shows that the computation times as well as the total branch and bound effort spent on optimizing the 0–1 linear programming problems are extremely low. To optimize the (previously unsolved) problem GRO48, a total of 18 nodes had to be generated and the entire execution time took 5 seconds of CPU time. The 'gap' between the linear program value of the corresponding TSP output and the optimal tour is 14.94, i.e. roughly 0.3% of the optimal tour length, and this fact together with the high cutting power of the facet-inducing subtour elimination and comb constraints (of which a total of 38 was generated) is responsible for the low computation time. The same can be said about virtually every other problem. The gap between the linear programming value LPVALUE and the optimum tour for KRO125 is the largest gap of these 100-city problems and its value RATIO (see Table 9.5) is one of the lowest ratios of the table. Why this is so, one can only guess; but it is clearly indicative of the comparatively longer time it took to solve KRO125. Still, only two iterations were required and (absolutely speaking) the total number of nodes in the respective branching trees is still very small. For the problems GRO120 and KNU121, the 0–1 solutions found at the end of the intermediate

iterations always had the same objective function value as the optimal tour, so the additional iterations were needed to exclude these solutions with subtours.

LIN318A and LIN318B are both runs of the 318-city problem LIN318. The first run LIN318A was made with the output from the cutting plane method (TSP) using the suboptimal tour of length 41,349 found by Padberg & Hong [1980]. After three iterations, the program halted and the optimal tour of length 41,345 was found. Using Stirling's approximation formula for $n!$, the optimal tour displayed in Figure 9.8 is the (possibly unique) tour (having one arc fixed) from among 10^{655} tours that are possible among 318 points and have one arc fixed. Assuming that one could possibly enumerate 10^9 solutions (tours) per second on a computer it would thus take roughly 10^{639} years of computing to establish the optimality of this tour by exhaustive enumeration. Solving the TSP output to optimality took less than 6 minutes of CPU time. The run LIN318B was essentially made to validate the previous run; in a way, it was sort of a check of internal consistency. First, the TSP code was rerun with the optimal tour as the starting solution.

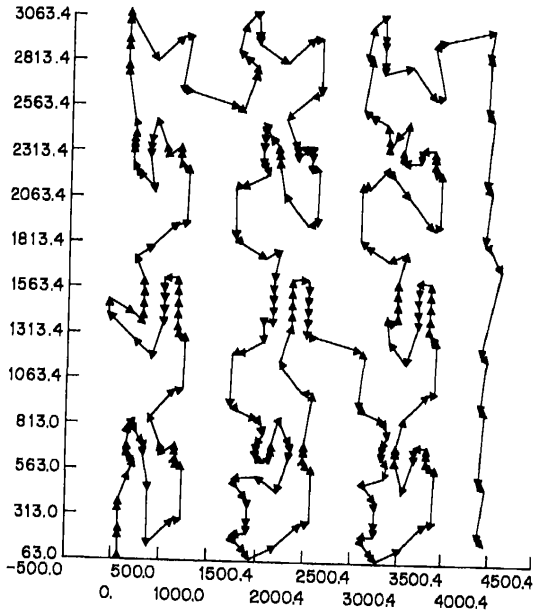

Figure 9.8 Solution of a 318-city problem

The results of this run are displayed in Table 9.7 in the row labeled LIN318B. As a result of the (slightly) smaller gap, the TSP code fixed more variables than previously and the problem generated had 495 rows and 1,144 variables. Again, within less than 6 minutes of CPU time, the optimal solution of length 41,345 was obtained and thus the optimal solution verified.

LIN318C is a run of LIN318 using a factor of 10 in the original coordinates. (Thus the distances are calculated on the basis of $(10x_i, 10y_i)$ rather than on the basis of (x_i, y_i) where $i = 1, \ldots, 318$.) This run was executed to test the issue of *precision* in the calculation of the Euclidean distances referred to in Section 4.2. While both the computation times and the number of nodes increased markedly with the increased number of digits to be processed, the last row of Table 9.7 exhibits substantial similarity to the run LIN318B. More importantly, increasing the precision in the distance table did not alter the optimal tour displayed in Figure 9.8.

The problem with 318 cities should not be the end of the story. The codes we have reported here have used only some of the heuristics described above. Currently, new linear programming based cutting plane procedures are being developed which contain exact subroutines for the identification of subtour elimination and 2-matching constraints as well as several new heuristics and new implementation details to speed up various communication procedures between subroutines and to overcome certain problems with respect to space. These methods may lead to another jump in the range of solvable problem sizes.

Designing and programming such TSP codes is by no means easy. But it seems to be worth the effort, since we know of quite a number of real-world TSPs of size 500 to 6,000 cities which are waiting to be solved. The real world – of course – remains to be a permanent challenge for combinatorial optimizers.

The Traveling Salesman Problem
Edited by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys

10

Branch and bound methods

E. Balas
Carnegie-Mellon University, Pittsburgh

P. Toth
Università di Bologna

1 INTRODUCTION

The origins of the branch and bound idea go back to the work of Dantzig, Fulkerson & Johnson [1954, 1959] on the TSP. The first full-fledged

The research of the first author was supported by Grant ECS-8205425 of the U.S. National Science Foundation and by Contract N00014-75-C-0621 NR 047-048 with the U.S. Office of Naval Research; and that of the second author by the Consiglio Nazionale delle Ricerche of Italy.