DIMACS Technical Report 95-05
March 1995

FINDING CUTS IN THE TSP
(A preliminary report)

by

David Applegate[1]
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Robert Bixby[2]
Dept. of Comp. & Applied Math.
Rice University
Houston, Texas 77005

Vašek Chvátal[3]
Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

William Cook[4]
Bellcore
Morristown, New Jersey 07960

[1]Permanent Member
[2]Supported by Center for Research in Parallel Computing, Rice University and by NSF grants CCR-881594 and CCR-9407142
[3]Permanent Member
[4]Permanent Member

ABSTRACT

TSPLIB is Gerhard Reinelt's library of some hundred instances of the traveling salesman problem. Some of these instances arise from drilling holes in printed circuit boards; others arise from X-ray crystallography; yet others have been constructed artificially. None of them (with a single exception) is contrived to be hard and none of them is contrived to be easy; their sizes range from 17 to 85,900 cities; some of them have been solved and others have not.

We have solved twenty previously unsolved problems from the TSPLIB. One of them is the problem with 225 cities that was contrived to be hard; the sizes of the remaining nineteen range from 1,000 to 7,397 cities.

Like all the successful computer programs for solving the TSP, our computer program follows the scheme designed by George Dantzig, Ray Fulkerson, and Selmer Johnson in the early nineteen-fifties. The purpose of this preliminary report is to describe *some* of our innovations in implementing the Dantzig-Fulkerson-Johnson scheme; we are planning to write up a more comprehensive account of our work soon.

1 INTRODUCTION

The traveling salesman problem, or TSP for short, is easy to state: given a finite number of "cities" along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities and returning to your starting point. (The travel costs are symmetric in the sense that traveling from city X to city Y costs just as much as traveling from Y to X; the "way of visiting all the cities" is simply the order in which the cities are visited.) To put it differently, the data consist of integer weights assigned to the edges of a finite complete graph; the objective is to find a hamiltonian cycle (that is, a cycle passing through all the vertices) of the minimum total weight. In this context, hamiltonian cycles are commonly called *tours*.

The origins of the TSP are obscure. In the 1920's, the mathematician/economist Karl Menger publicized it among his colleagues in Vienna; in the 1930's, the problem reappeared in the mathematical circles of Princeton; in the late 1940's, the mathematician Merill Flood popularized it among his colleagues at the RAND Corporation. Eventually, the TSP gained notoriety as the prototype of a hard problem in combinatorial optimization: examining the tours one by one is out of the question because of their large number, and no other idea was on the horizon for a long time. (A close relative of the TSP is the *minimum spanning tree* problem, where a minimum-weight spanning tree rather than a minimum-weight spanning cycle is sought. This apparently minor modification makes the new problem far easier: the first efficient algorithm for solving the minimum spanning tree problem was proposed as early as 1926 [3]. Just as the TSP is the prototype of a *hard* combinatorial optimization problem, the minimum spanning tree problem is the prototype of an *easy* combinatorial optimization problem. From this perspective, arguing the difficulty of the TSP by harping on the large number of tours is not entirely convincing: the number of spanning trees in a complete graph is much larger than the number of tours.)

A breakthrough came in 1954, when George Dantzig, Ray Fulkerson, and Selmer Johnson [7] published a description of a method for solving the TSP and illustrated the power of this method by solving an instance with 49 cities, an impressive size at that time. Riding the wave of excitement over the numerous applications of the simplex method (designed by George Dantzig in 1947), they attacked the salesman with linear programming as follows.

Each TSP instance with n cities can be specified as a vector of length $n(n-1)/2$ (whose components, indexed by the edges of the complete graph, specify the costs) and each tour through the n cities can be represented as its incidence vector of length $n(n-1)/2$ (with each component set at 1 if the corresponding edge is a part of the tour, and set at 0 otherwise); if c^T denotes the cost vector (thought of as a row vector) and if $\mathcal{S}$ denotes the set of the incidence vectors (thought of as column vectors) of all the tours, then the problem is to

$$\text{minimize } c^T x \text{ subject to } x \in \mathcal{S}. \tag{1.1}$$

Like the man searching for his lost wallet not in the dark alley where he actually dropped it, but under a streetlamp where he can see, Dantzig, Fulkerson and Johnson begin not with the problem they *want to* solve, but with a related problem they *can* solve,

$$\text{minimize } c^T x \text{ subject to } Ax \le b \tag{1.2}$$

with some suitably chosen system $Ax \le b$ of linear inequalities satisfied by all x in $\mathcal{S}$: solving linear programming problems such as (1.2) is precisely what the simplex method is for.

Since (1.2) is a *relaxation* of (1.1) in the sense that every feasible solution of (1.1) is a feasible solution of (1.2), the optimal value of (1.2) provides a lower bound on the optimal value of (1.1). The ground-breaking idea of Dantzig, Fulkerson, and Johnson was that solving (1.2) can help with solving (1.1) in a far more substantial way than just by providing a lower bound: having satisfied oneself that the wallet is not under the streetlamp, one can pick the streetlamp up and bring it a little closer to the place where the wallet was lost. It is a characteristic feature of the simplex method that the optimal solution x^* it finds is an extreme point of the polyhedron defined by $Ax \le b$; in particular, if x^* is not one of the points in $\mathcal{S}$ then it lies outside the convex hull of $\mathcal{S}$. In that case, x^* can be separated from $\mathcal{S}$ by a hyperplane: some linear inequality is satisfied by all the points in $\mathcal{S}$ and violated by x^*. Such an inequality is called a *cutting plane* or simply a *cut*. Having found a cut, one can add it to the system $Ax \le b$, solve the resulting tighter relaxation by the simplex method, and iterate this process until a relaxation (1.2) with an optimal solution in $\mathcal{S}$ is found.

We shall illustrate the Dantzig-Fulkerson-Johnson method on the Dantzig-Fulkerson-Johnson example, the 49-city problem. This instance of the TSP was created by picking one city from each of the 48 states of the U.S.A. (Alaska and Hawaii became states only in 1959) and adding Washington, D.C.; the costs of travel between different cities were defined as road distances taken from an atlas. (In the actual computations, each distance of d miles was replaced by $(d-11)/17$ rounded up to the nearest integer, so that each of the resulting numbers could be stored as a single byte.) Rather than solving this 49-city problem, Dantzig, Fulkerson and Johnson solved the 42-city problem obtained by removing Baltimore, Wilmington, Philadelphia, Newark, New York, Hartford, and Providence. As it turned out, an optimal tour of the 42 cities used the edge from Washington, D.C. to Boston; since the shortest route between these two cities passes through the seven removed cities, this solution of the 42-city problem yields a solution of the 49-city problem.

Here, and throughout the paper, we shall conform to the following conventions. The symbol V is reserved for the set of cities; each edge of the complete graph with vertex-set V is simply a two-point subset of V; if x is a vector whose components are indexed by the edges of this complete graph, then x_e or $x(e)$ denotes the component indexed by e; we write

$$x(S) = \sum(x_e : e \subseteq S)$$

for every set S of cities and

$$x(S,T) = \sum(x_e : e \cap S \ne \emptyset,\ e \cap T \ne \emptyset)$$

for every choice of disjoint sets S, T of cities. By the *graph of* x, we shall mean the graph whose vertex-set is V, and whose edges are all the e with $x_e > 0$.

Trivially, each x in $\mathcal{S}$ satisifies

$$0 \leq x_e \leq 1 \qquad \text{for all edges } e \tag{1.3}$$

and

$$x(\{v\}, V - \{v\}) = 2 \qquad \text{for all cities } v, \tag{1.4}$$

and so one can always use the system (1.3), (1.4) as the initial choice of the $Ax \leq b$ in (1.2).

In the 42-city problem, the graph of the optimal solution x^* of this initital relaxation is disconnected: one of its two components has vertices $1, 2, 41, 42$ and the other component has vertices $3, 4, \ldots, 40$. This structural fault makes the first cut obvious: since every tour must cross every demarcation line separating V into two nonempty parts at least twice, every x in $\mathcal{S}$ satisfies

$$x(S, V - S) \geq 2 \tag{1.5}$$

for all nonempty proper subsets S of V; however, x^* in place of x violates (1.5) with $S = \{1, 2, 41, 42\}$. Constraints (1.5), called "loop conditions" by Dantzig, Fulkerson, and Johnson, are nowadays called *subtour elimination constraints* or simply *subtour constraints*: from the set of integer solutions of (1.3), (1.4), they eliminate incidence vectors of disconnected graphs (consisting of two or more vertex-disjoint "subtours") and leave only the incidence vectors of tours.

The next two iterations are similar: the graphs of x^* are disconnected and we add two more subtour constraints (one with $S = \{3, 4, \ldots, 9\}$, the other with $S = \{24, 25, 26, 27\}$). Then the graph of x^* becomes connected but not 2-connected: removal of city 18 splits it into two connected components, one with vertices $13, 14, \ldots, 17$ and the other with vertices $19, 20, \ldots, 42, 1, 2, \ldots, 12$. Again, this structural fault in the graph points out a violated subtour constraint: more generally, if the removal of a single city, say w, splits the rest of the graph into connected components with vertex-sets $S_1, S_2, \ldots, S_k$ ($k \geq 2$) then trivially $x^*(S_i, V - S_i) = x^*(S_i, \{w\}) \leq 1$ for at least one i. We add the subtour constraint with $S = \{13, 14, \ldots, 17\}$ and continue through three similar iterations (adding subtour constraints with $S = \{10, 11, 12\}$, $S = \{11, 12, \ldots, 23\}$, and $S = \{13, 14, \ldots, 23\}$) until we obtain a relaxation (1.2), whose optimal solution x^* is shown in Fig. 1.1: the solid edges e carry $x_e = 1$, the dashed edges e carry $x_e = 1/2$, and the long path 28-29-30- …-41-42-1-2- …-8-9-10 consists entirely of solid edges. The graph of this x^* is 2-connected; no violated subtour constraints are apparent; in fact (as we shall observe in the next section), there are none.

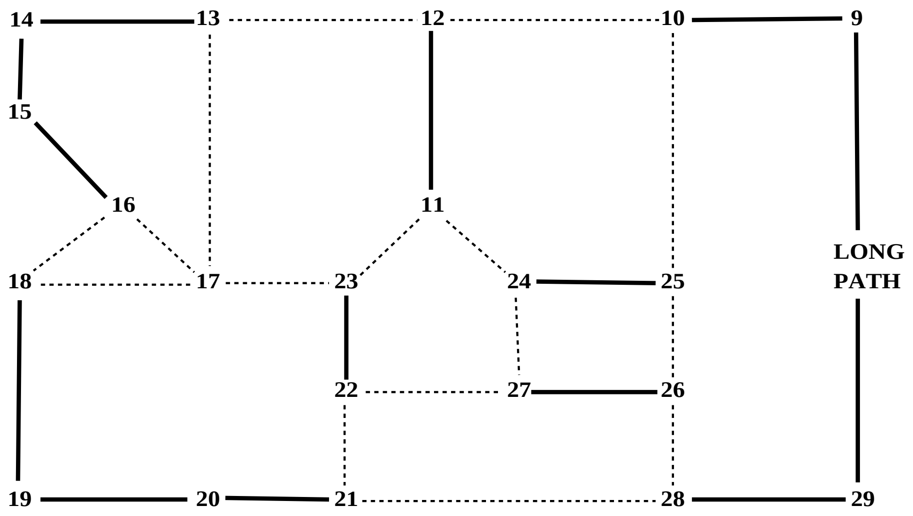

Fig.1.1: What is wrong with this vector?

Now Dantzig, Fulkerson, and Johnson add two more linear inequalities satisfied by all x in $\mathcal{S}$ and say in a footnote, "We are indebted to I. Glicksberg of Rand for pointing out relations of this kind to us". These two inequalities read

$$x(\{15,16,18,19\}) + 2x_{\{14,15\}} + x_{\{16,17\}} + x_{\{19,20\}} \leq 6 \tag{1.6}$$

(actually, this constraint is presented in [7] as our (1.6) minus the sum of the three equations $x(\{v\}, V - \{v\}) = 2$ with $v = 15, 16, 19$) and

$$\sum a_e x_e \leq 42 \tag{1.7}$$

with $a_{\{22,23\}} = 2$ and $a_e = 1$ for all other e except that $a_e = 0$ when (i) $e = \{25,26\}$, or (ii) $x^*_e = 0$ and $|e \cap \{10,11,\ldots,28\}| = 1$, or (iii) $x^*_e = 0$ and $|e \cap \{10,21,25,26,27,28\}| \geq 1$. It is easy to see that each x in $\mathcal{S}$ satisfies (1.6); in fact, each x in $\mathcal{S}$ satisfies

$$x(\{15,16,18,19\}) + x_{\{14,15\}} + x_{\{16,17\}} + x_{\{19,20\}} \leq 5\,; \tag{1.8}$$

the sum of (1.8) and the trivial $x_{\{14,15\}} \leq 1$ is (1.6). To check that each x in $\mathcal{S}$ satisfies (1.7), assume the contrary: the incidence vector of some tour T violates (1.7). Obviously, T has to use the edge $\{22,23\}$ and has to avoid all the edges e with $a_e = 0$. Since $\{9,10\}$ and $\{28,29\}$ are the only two edges e with $a_e \neq 0$ and precisely one endpoint in $\{10,11,\ldots,28\}$, they have to be used by T; since $\{10,25\}$ and $\{24,25\}$ are the only two edges e with $a_e \neq 0$ and one endpoint equal to 25, they have to be used by T; since $\{26,27\}$ and $\{26,28\}$ are the only two edges e with $a_e \neq 0$ and one endpoint equal to 26, they have to be used by T. Since T uses $\{26, 28\}$ and $\{28, 29\}$, it must avoid $\{21, 28\}$; since $\{20, 21\}$ and $\{21, 22\}$ are the only remaining edges e with $a_e \neq 0$ and one endpoint equal to 21, they have to be used by T. Since T uses $\{21, 22\}$ and $\{22, 23\}$, it must avoid $\{22, 27\}$; since $\{24, 27\}$ and $\{26, 27\}$ are the only remaining edges e with $a_e \neq 0$ and one endpoint equal to 27, they have to be used by T. But then T contains a cycle on $1, 2, \ldots, 10, 25, 24, 27, 26, 28, 29, \ldots, 42$, a contradiction.

By adding constraints (1.6) and (1.7) to the previous relaxation of (1.1), Dantzig, Fulkerson and Johnson finally put the streetlamp next to the wallet: they obtained a relaxation, whose optimal solution is the incidence vector of a tour (passing through the 42 cities in the order of their labels). And that was the end of the 49-city problem.

The influence of this work reached far beyond the narrow confines of the TSP.

On the one hand, the method can be used to attack any problem of the form (1.1), regardless of the particular choice of $\mathcal{S}$, as long as there is an efficient algorithm to recognize points of $\mathcal{S}$ (we have to know a good thing when we see it returned as an optimal solution of a relaxation (1.2)). Many problems in combinatorial optimization have this form: in the maximum clique problem, $\mathcal{S}$ consists of the incidence vectors of all cliques in the input graph G (the components of these vectors are indexed by the vertices of G); in the maximum cut problem, $\mathcal{S}$ consists of the incidence vectors of all edge-cuts in the input graph G (the components of these vectors are indexed by the edges of G); and so on. In each of these particular examples, the challenge is to come up with linear inequalities satisfied by all the points in $\mathcal{S}$ and violated by the optimal solution x^* of the current relaxation (1.2). To meet this challenge, one often begins with common sense to establish some combinatorial property of $\mathcal{S}$ and only then (as we have seen in the 42-city example) one expresses this property in terms of linear inequalities. This line of attack led to the development of the flourishing field of *polyhedral combinatorics.*

Second, the method can be used to attack any integer linear programming problem. (To cast the TSP in this form, note that its $\mathcal{S}$ consists of all integer solutions of (1.3), (1.4), (1.5).) Again, the challenge is to come up with linear inequalities satisfied by all the integer feasible solutions of the current relaxation (1.2) and violated by its optimal solution x^*: can I. Glicksberg's ingenuity be replaced by an automatic procedure to generate cutting planes? Ralph Gomory [8], [9], [10] answered this challenge with breathtaking elegance by his design of *cutting-plane algorithms.*

In this way, the work of Dantzig, Fulkerson, and Johnson became the prototype of two different methodologies: polyhedral combinatorics in combinatorial optimization and cutting-plane algorithms in integer linear programming.

This seems to be the reason why the TSP is so popular. Arguing that it is popular because it arises from practical applications would be hard: even though variations on the TSP theme come up in practice relatively often, the theme in its pure form appears only rarely. Two of its sources are (a) drilling holes in printed circuit boards, where the time spent on moving the drill through a sequence of prescribed positions is to be minimized, and (b) X-ray crystallography, where the time spent on moving the diffractometer through a sequence of prescribed angles is to be minimized [1]. A number of TSP instances arising from these sources, along with instances created artificially in the Dantzig-Fulkerson-Johnson tradition (by taking 120 cities in Germany, or 532 cities in the U.S.A., or 666 cities all over the world, or . . .), have been used for years as standard test problems for computer algorithms; recently, Gerhard Reinelt [27, 28] collected around a hundred of them, with sizes ranging from 17 to 85,900 cities, in a library called TSPLIB.

Writing computer programs to solve TSPLIB problems can hardly be classified as *applied* work. The technology of manufacturing printed circuit boards has changed and the drilling problems from the TSPLIB are no longer of interest to the industry. Similarly, development of multiplex cameras is making the TSP problems of X-ray crystallography obsolete; besides, research laboratories have never considered large investments of computer time for the small gain of minimizing the time spent by moving the diffractometer. Furthermore, even if there were a client with a genuine need to solve TSP problems, such a client would most likely be satisfied with *nearly* optimal tours. Finding nearly optimal tours even in fairly large TSPLIB problems is a relatively easy task: good implementations of the Lin-Kernighan heuristic [19] and its refinements work like a charm. Most of the computer time spent on solving TSPLIB problems goes into *proving* that a tour is optimal, a fact of negligible interest to the hypothetical client.

Writing computer programs to solve TSPLIB problems can hardly be classified as *theoretical* work, either. A prize offered by the RAND corporation for a significant theorem bearing on the TSP was never awarded; Dantzig, Fulkerson, and Johnson close their seminal paper with the modest disclaimer " It is clear that we have left unanswered practically any question one might pose of a theoretical nature concerning the traveling-salesman problem . . . ". All the successful computer programs for solving TSP problems follow the Dantzig-Fulkerson-Johnson scheme; improvements consist only of better ways of finding cuts and better handling of the large linear programming relaxations. (Having a faster computer helps, too.) It might be argued that the most significant theorem bearing on the TSP is one pointing in the opposite direction: Richard Karp, Eugene Lawler, and Robert Tarjan [18] proved that the decision version of the TSP is an $\mathcal{NP}$-complete problem. (In quite a few of the TSPLIB instances, the cities are points in the plane and the cost function is some natural metric; the decision version of the TSP remains $\mathcal{NP}$-complete even when its inputs are restricted to such special instances: see [17].)

Writing computer programs to solve TSPLIB problems could be classified as a *sport*, where each new record is established by solving at least one previously unsolved instance. A few milestone records are shown in Table 1.1; the suffix of each problem name, as given in TSPLIB, specifies the number of cities.

PROBLEM	SOLVED BY
`gr120`	Grötschel [11]
`lin318`	Crowder and Padberg [6]
`pcb442`	Grötschel and Holland [12]
`att532`	Padberg and Rinaldi [24]
`gr666`	Grötschel and Holland [12]
`pr1002`	Padberg and Rinaldi [24]
`pr2392`	Padberg and Rinaldi [24]

Table 1.1: The TSP Olympics

Progress in this discipline is made by solving harder and harder problems; "harder" is by no means synonymous with "larger". On the one hand, the fact that recognizing upper bounds on the optimal tour length is an $\mathcal{NP}$-complete problem suggests (subject to the article of faith that $\mathcal{P} \neq \mathcal{NP}$) the existence of relatively small TSP instances guaranteed to require prohibitively large amounts of time by *any* algorithm; for relatively small TSP instances guaranteed to require prohibitively large amounts of time by any of the algorithms *in current use*, see [5]. On the other hand, any reasonable TSP algorithm should make an easy job of solving an instance created by taking the euclidean distances between 10000 randomly chosen points on a circle. (Of course, one could always argue that the algorithm never made use of the special structure of this instance as long as it has not been explicitly told to do so.) Problems from the TSPLIB are not like either of these two extremes: they (with the exception of `ts225`) are not contrived to be hard and they are not contrived to be easy. Still, they are subject to statistical variations of difficulty.

What does one mean by saying that a particular TSP instance is difficult? A really huge instance can overwhelm the machine simply by its sheer size; however, there are ways in which even relatively small instances can turn nasty. Success of the Dantzig-Fulkerson-Johnson method hinges not on the ability to find cuts (Gomory taught us how to always do that) but on the ability to find cuts that are strong enough, so that their insertion into the current relaxation brings about significant progress towards solving the instance. This progress is often estimated by the increase in the optimal value of the relaxation; as more and more cuts are added, these increases tend to get smaller and smaller. If these diminishing returns get below an acceptable threshold, then it may be preferable to *branch* by splitting the problem into two subproblems. Typically, this is done by first picking an edge, say e, and then looking separately for (i) the cheapest tour that uses e and (ii) the cheapest tour that avoids e. Again, each of the two subproblems is attacked by solving its linear programming relaxation (with constraint $x_e = 1$ added to subproblem (i) and constraint $x_e = 0$ added to subproblem (ii)) and adding new cuts (that is, linear inequalities satisfied by every incidence vector of a tour and violated by an optimal solution of the current relaxation of the subproblem) to the old; again, failure to find sufficiently many sufficiently strong cuts may force splitting one or both of the two subproblems into sub-subproblems, and so on. In the resulting binary tree of subproblems, some branches may be pruned off without exploring the subtrees they lead to: from the outset, one stores the best currently known tour and uses its cost as a *bound* on the cost of an optimal tour. If the optimal value of a subproblem's relaxation is at least this bound, then no tour in the subproblem has a chance of beating the current incumbent, and so the subproblem can be abandoned at once. (In addition, a subproblem can be abandoned if its relaxation has no feasible solutions.) This scheme, apparently used first by Miliotis [20] and later, in a different context, by Grötschel, Jünger, and Reinelt [13], is known as *branch-and-cut* [24].

In a way, the number of nodes in the branch-and-cut tree of subproblems (for a fixed method of finding cuts and a fixed heuristic to find the initial incumbent tour) could be taken as a measure of the *inherent* difficulty of an instance, independent of its size: its internal nodes are in a one-to-one correspondence with subproblems so difficult that, given any of them, the algorithm resorted to brute force and simply split it into two sub-subproblems. In this perspective, Table 1.2 illustrates the point that larger does not always mean harder: curiously, the number of nodes in the branch-and-cut tree decreases here as the problems get bigger. (The four problems are named ATT532, GH666, TK1002, and TK2392 in [26]. Padberg and Rinaldi apparently do not count the root as one of the nodes of the branch-and-cut tree; we do.)

PROBLEM	NUMBER OF NODES IN THE BRANCH-AND-CUT TREE OF [26]
`att532`	107
`gr666`	21
`pr1002`	13
`pr2392`	3

Table 1.2: A monotonically decreasing function

Dantzig, Fulkerson, and Johnson showed a way to solve large instances of the TSP; all that came afterward is just icing on the cake. The purpose of the present paper is to describe some of the icing we have added on top of the previous layers. Our icing comes in five flavors:

(i) new ways of finding cuts,

(ii) new ways of handling the LP relaxations,

(iii) new ways of selecting an edge on which to branch,

(iv) new ways of finding an incumbent tour
(and so pruning the branch-and-cut tree more often),

(v) solving the problem in parallel on a network of UNIX workstations.

The purpose of this preliminary report is to describe a *subset* of (i); we are planning to write up a more comprehensive account of our work soon.

Our initial goal was to solve TSPLIB problem `pcb3038`. To attain this goal, we had armed ourselves with some sixty workstations and written, as well as we could, a computer program based on previously published work of others. The cut-finding techniques of our program included

- the Crowder-Padberg "shrinking procedure" [6],
- the separation algorithm for subtour constraints that solves $n-1$ max-flow min-cut problems
(see, for instance, Section 2.2 of [21]),
- the Padberg-Rao separation algorithm for blossom constraints [23],
- the Grötschel-Holland comb heuristics [12],
- the Padberg-Rinaldi comb and clique-tree heuristics [25],

as well as a number of our own innovations (not described in this report); our LP solver was CPLEX, which we modified in significant ways. In January 1992, we began our first run on `pcb3038`. As we monitored the growth of the branch-and-cut tree during a few weeks, it was becoming more and more obvious to us that, despite the considerable computing power at our disposal, we had no hope of ever solving the problem: the tree just kept on growing and growing and growing. We had to either come up with new tricks or give up.

We did come up with new tricks and solved the problem by the end of April. In retrospect, we are not sure which of these new tricks got us over the `pcb3038` hump. We believe it was not any single one of them, but a combination of three cut-finding techniques described in this report (in Sections 6, 7, and 8) and a certain time-consuming way to branch. Over the next couple of years, we added a few more improvements (including the necklace technique of Section 3) and solved additional problems from the TSPLIB. Among the previously unsolved problems that we have solved are `d1291, d1655, dsj1000, fl1400, fnl4461, nrw1379, pcb1173, pcb3038, pla7397, rl1304, rl1323, rl1889, ts225, u1060, u1432, u1817,` `u2152`, `u2319`, `vm1084`, and `vm1748`; certificates of the optimality of our solutions of `fnl4461`, `pcb3038`, `pla7397`, and `ts225` are available by anonymous ftp from

`netlib.att.com`

in the directory

`netlib/att/math/applegate/TSP` .

The remaining unsolved problems in TSPLIB-Version 1.2 are `brd14051`, `d2103`, `d18512`, `fl1577`, `fl3795`, `pla33810`, `pla85900`, `rl5915`, `rl5934`, and `rl11849`.

2 TIGHT SETS AND PQ-TREES

Throughout this paper, x will always denote a vector, with components indexed by the edges of the complete graph on vertex-set V, that satisfies

$$0 \leq x_e \leq 1 \qquad \text{for all edges } e \tag{2.1}$$

and

$$x(\{v\}, V - \{v\}) = 2 \qquad \text{for all cities } v. \tag{2.2}$$

The purpose of the present section is to introduce a certain *hierarchical decomposition* of V with respect to x; we shall use this decomposition again and again to find cuts.

This decomposition revolves around the notion of a *tight set*, by which we mean any set S of cities such that $x(S, V - S) = 2$. For reasons that will become apparent later, we find it useful to store as many tight sets as we can find without undue effort. To store these sets, we first choose an arbitrary city, call it `exterior`, and write $W = V - \{\texttt{exterior}\}$; since S is tight if and only if $V - S$ is tight, we lose no generality by storing only tight subsets of W.

A *PQ-tree* (introduced by Booth and Lueker [2]) is a rooted ordered tree with each internal node having at least two children and labeled either as a *P-node* or as a *Q-node*; it is customary to draw P-nodes as circles and Q-nodes as rectangles. For each node u of such a PQ-tree T, we let $D(u)$ denote the set of all leaves that are descendants of u; we let $\mathcal{B}(T)$ denote the family of

- all sets $D(u)$
  such that u is a node of T, and
- all sets $D(u_i) \cup D(u_{i+1}) \cup \ldots \cup D(u_j)$
  such that $u_1, u_2, \ldots, u_k$ in this order are the children of some Q-node of T and $1 \leq i < j \leq k$.

We say that a PQ-tree is *compatible* with x if the set of its leaves is W and all sets in $\mathcal{B}(T)$ are tight. (In these definitions, it is immaterial whether a node with precisely two children is labeled a P-node or a Q-node; Booth and Lueker consider it P-node; for our purpose, it will be more convenient to consider it a Q-node.) For example, the PQ-tree shown in Fig.2.1 is compatible with the vector x shown in Fig.1.1; there, we have chosen $\texttt{exterior} = 42$.

We are about to prove a theorem specifying a minor reason why we are interested in PQ-trees (the major reasons will come in subsequent sections). This theorem involves the intuitive notion of *shrinking* a set of cities. Formally, shrinking a subset S of V means replacing V with V/S defined as $(V - S) \cup \{\sigma\}$ for some new vertex σ (representing the shrunk S) and replacing x with x/S defined on the edges of the complete graph with vertex-set V/S by

$$x/S(\{\sigma, t\}) = x(S, t) \text{ for all } t \text{ in } V - S$$

and $x/S(t_1, t_2) = x(t_1, t_2)$ for all t_1, t_2 in $V - S$. (If S is tight then the resulting x/S satisfies (2.2) in place of x.) With each internal node u of a PQ-tree compatible with x, we associate the vector $x[u]$ that arises from x by shrinking $V - D(u)$ as well as each $D(v)$ such that v is a child of u.

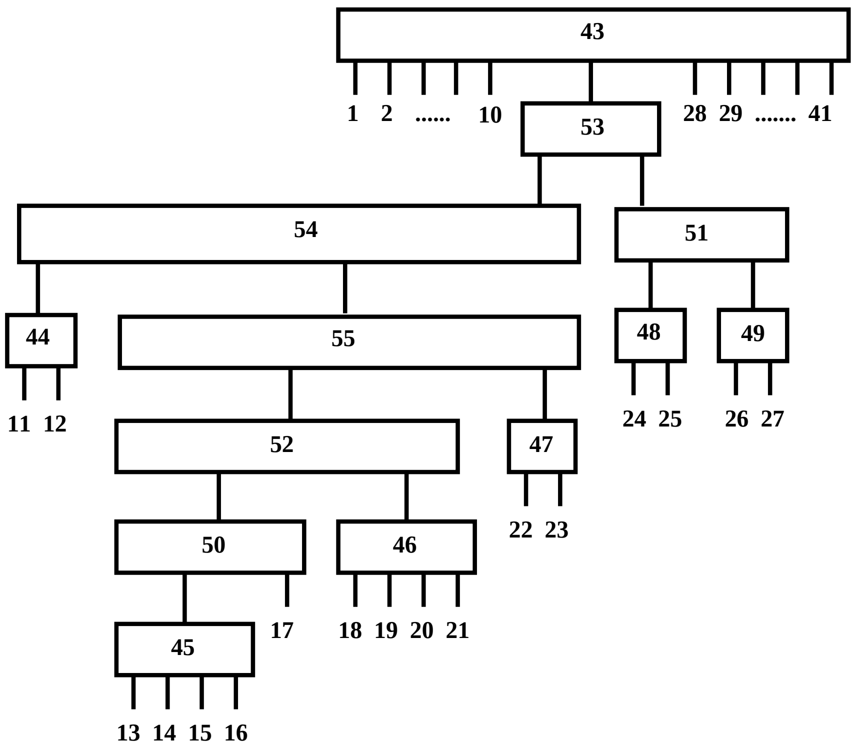

Figure 2.1: A PQ-tree compatible with the vector in Fig.1.1.

THEOREM 2.1 *For every PQ-tree T compatible with x, the following two conditions are equivalent:*

(i) x satisfies all subtour constraints,

(ii) $x[u]$ satisfies all subtour constraints for each P-node u of T. □

This is one reason why we are interested in PQ-trees: if a PQ-tree compatible with x is available and if each P-node of this tree has a relatively small number of children, then the problem of finding subtour constraints violated by x reduces to smaller subproblems. (For instance, since the PQ-tree in Fig. 2.1 has no P-nodes at all, Theorem 2.1 guarantees immediately that the vector in Fig. 1.1 satisfies all subtour constraints.)

It will be convenient to set a part of the argument used to prove Theorem 2.1 on its own as a lemma. We call a subset S of V *bad* if S is a nonempty proper subset of V and $x(S, V-S) < 2$.

LEMMA 2.1 *If R is tight, if S is bad, and if $R \cap S \neq \emptyset$, $R \cup S \neq V$, then at least one of $R \cap S$ and $R \cup S$ is bad.*

PROOF. Writing $A = R - S$, $B = R \cap S$, $C = S - R$, and $D = V - (A \cup B \cup C)$, observe that
$x(B, V-B) + x(D, V-D) = x(R, V-R) + x(S, V-S) - 2x(A, C) < 4.$ □

PROOF OF THEOREM 2.1. The "if" part: Assume that x does not satisfy all subtour constraints: some subset of W is bad. Let u be the lowest node in the tree such that $D(u)$ contains a bad set and call this bad set S. Now scan all the children v of u such that $S \cap D(v) \neq \emptyset$ one by one and, for each of these children v, replace S with $S \cup D(v)$. Since $S \cap D(v)$ is not bad (by our choice of u), Lemma 2.1 with $R = D(v)$ guarantees that S remains bad throughout this iterative process; in the end, S is the union of sets $D(v)$ with v ranging through some nonempty set of children of u. Trivially, the counterpart of S (obtained by shrinking $V - D(u)$ as well as each $D(v)$ such that v is a child of u) is bad for $x[u]$; since $x[w]$ is the incidence vector of a cycle whenever w is a Q-node, u must be a P-node.

The "only if" part is trivial. □

Theorem 2.1 facilitates the search for subtour constraints violated by x only if a PQ-tree compatible with x is available and only if each P-node of this tree has a relatively small number of children. Figure 2.2 describes a function `crude` that, given a vector x (satisfying (2.1), (2.2) as declared at the beginning of this section), either produces a PQ-tree compatible with x or returns a bad set. (This description is not quite deterministic: there may be quite a few choices of `F` in quite a few iterations.) The PQ-tree produced by `crude` is special: except possibly for a single node `current`, all of its internal nodes are Q-nodes. Hence Theorem 2.1 reduces in this case to the statement that

$$x \text{ satisfies all subtour constraints if and only if } x[\texttt{current}] \text{ does.} \qquad (2.3)$$

Removal of the statements that construct the PQ-tree reduces `crude` to the "shrinking procedure" of Crowder and Padberg [6], who use it along with (2.3) to find bad sets. Given a vector x such that $x_e < 1$ for all edges e, `crude` finds no bad sets whatsoever and produces the trivial PQ-tree whose root, the P-node `current`, is the common parent of all the cities in W; now (2.3) is a tautology. Nevertheless, the performance of `crude` is often adequate:

for instance (see Figure 2.3), it produced the PQ-tree in Fig. 2.1. We shall describe a way of refining crude PQ-trees in Section 4.

```
create a P-node root, whose children are all the cities except exterior;
current=root;

while there is an edge $e$ with $x_e = 1$
do $G$ = graph consisting of vertex-set $V$ and all the edges $e$ with $x_e = 1$;
   $F$ = arbitrary component of $G$ that has at least two vertices;

   if $F$ is a path, say, $w_1 w_2 \dots w_k$
   then shrink $F$;
        if shrinking $F$ produced an edge $uv$ with $x_{uv} > 1$
        then return the bad set $D(u) \cup D(v)$;
        else if $F$ does not include exterior
             then create a Q-node new with children $w_1, w_2, \dots, w_k$
                  on the list of children of current, replace $w_1, w_2, \dots, w_k$ with new;
             else $S$ = the set of all children of current that are not in $F$;
                  create a P-node new, whose children are the vertices in $S$;
                  $i$ = the subscript for which exterior=$w_i$;
                  make current into a Q-node with children
                       $w_{i-1}, w_{i-2}, \dots, w_1,$ new$, w_k, w_{k-1}, \dots, w_{i+1}$;
                  exterior = the shrunk $F$;
                  current = new;
             end
        end
   else (now $F$ is a cycle)
        if $F$ passes through all the vertices of $V$
        then make current into a Q-node with children ordered as in $F$;
             return;
        else return the bad set $\cup_{w \in F} D(w)$;
        end
   end

end
```

Figure 2.2: Function crude.

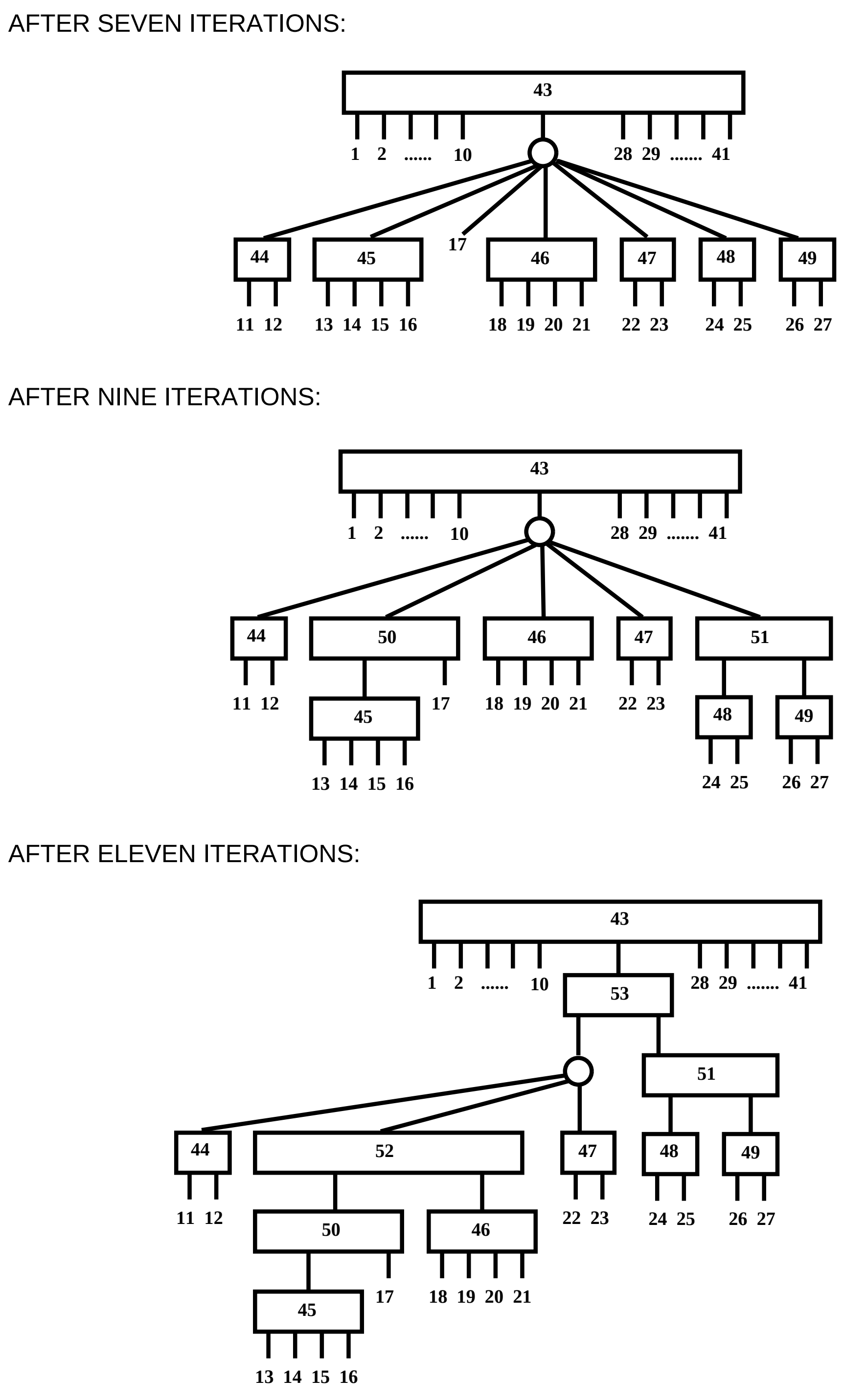

Figure 2.3: Function `crude()` working away on Figure 1.1.

3 FINDING CUTS: COMBS, DOMINOS, AND NECKLACES

If $T_1, T_2, \ldots, T_{2k+1}$ with $k \geq 1$ are pairwise disjoint subsets of V and if a subset H of V meets every T_i but does not contain any, then every incidence vector x of a tour satisfies

$$x(H) + \sum_{i=1}^{2k+1} x(T_i) \;\leq\; (|H|-1) + \sum_{i=1}^{2k+1} (|T_i|-1) - k. \tag{3.1}$$

(To see this, define c by $x(H) = |H| - c$. We may assume that $c \leq k$, since otherwise (3.1) holds trivially. Since the subgraph of the tour induced by H consists of c vertex-disjoint paths, there are at most $2c$ subscripts i such that the subgraph of the tour induced by T_i is a single path, and so $\sum x(T_i) \leq \sum(|T_i| - 1) - (2k + 1 - 2c)$.) Inequalities (3.1) are known as *comb inequalities.* The name comes from Chvátal [4], who introduced a variant of (3.1) with $T_1, T_2, \ldots, T_{2k+1}$ not required to be pairwise disjoint but each T_i restricted to meet H in precisely one point. The present theme is due to Grötschel and Padberg [14, 15], who have shown that it properly subsumes the original variation; we follow them in referring to H as the *handle* of the comb and to $T_1, T_2, \ldots, T_{2k+1}$ as its *teeth.* The purpose of this section is to describe first a method of finding comb inequalities violated by x in such a way that all T_i, $T_i \cap H$, and $T_i - H$ are tight, and then a heuristic extension of this method that may find additional comb inequalities violated by x.

By a *domino*, we mean any bipartite graph

- whose vertex-set is $A \cup B$ such that
  A, B are disjoint subsets of V and all three of A, B, $A \cup B$ are tight, and
- whose edge-set consists of all e
  with one endpoint in A, the other endpoint in B, and $x_e > 0$.

An *edge-cut* in a graph G is the set of all edges that, for some set H of vertices, have one endpoint in H and other endpoint outside H; throughout this section, we let G denote the graph of x. By a *cutter*, we mean any set of dominos such that

(i) the number of dominos in this set is odd,

(ii) the union of their edge-sets is an edge-cut,

(iii) their vertex-sets are pairwise disjoint.

Given a cutter $\{D_1, D_2, \ldots, D_{2k+1}\}$ and given a partition of the vertex-set of each D_i into tight sets A_i, B_i, we can always find a comb inequality violated by x; here is how. Property (ii) allows us to find a set H of vertices such that an edge of G has one endpoint in H and other endpoint outside H if and only if it is an edge of some D_i. Writing $T_i = A_i \cup B_i$, note that (since every domino has at least one edge) H meets every T_i but does not contain any; in addition, note that $x(H, V-H) = \sum x(A_i, B_i) = 2k+1$, which implies $x(H) = (|H|-0.5)-k$ by virtue of (2.2). Hence (3.1) is a comb inequality violated by x.

For sets of dominos that satisfy (iii), conditions (i) and (ii) amount to a system of linear congruences modulo two; we are about to explain how and why. Let $\mathcal{D}^*$ denote any set of dominos (viewed as the incidence vectors of their edge-sets whenever convenient). Now each subset $\mathcal{D}$ of $\mathcal{D}^*$ can be specified by its incidence vector s, whose components are indexed by the members of $\mathcal{D}^*$ (if d is the incidence vector of some domino in $\mathcal{D}$ then $s_d = 1$; else $s_d = 0$). In these terms, $\mathcal{D}$ has property (i) if and only if the sum of all the components of s is odd; with e standing for the vector of the same length as s and with all components equal to 1, this condition can be stated as

$$e^T s \equiv 1 \bmod 2. \tag{3.2}$$

As for (ii), it is well known and easy to prove that a set of edges is an edge-cut if and only if it meets each cycle in an even number of edges. Let $\mathcal{C}$ denote the set of all the incidence vectors of the edge-sets of cycles in G; now a set of edges is an edge-cut if and only its incidence vector y satisfies

$$c^T y \equiv 0 \bmod 2 \quad \text{for all } c \text{ in } \mathcal{C}. \tag{3.3}$$

If T is a spanning tree of G then for every edge e in $G - T$ there is precisely one cycle in $T + e$; the set of all such cycles, one for each e in $G - T$, is called a *fundamental set of cycles* (relative to T). Let $\mathcal{C}_0$ denote the set of all the incidence vectors of the edge-sets of cycles in some fundamental set. It is well known and easy to prove that each c in $\mathcal{C}$ is the sum modulo two of some subset of $\mathcal{C}_0$; hence the system (3.3) and its subsystem

$$c^T y \equiv 0 \bmod 2 \quad \text{for all } c \text{ in } \mathcal{C}_0 \tag{3.4}$$

have the same solutions. If $\mathcal{D}$ satisfies (iii) then the incidence vector of the union of the edge-sets of all its dominos is $\sum s_d d$; in that case, $\mathcal{D}$ has property (ii) if and only if

$$\sum_{d \in \mathcal{D}^*} s_d c^T d \equiv 0 \bmod 2 \quad \text{for all } c \text{ in } \mathcal{C}_0. \tag{3.5}$$

System (3.5) can be recorded in matrix notation as

$$As \equiv 0 \bmod 2; \tag{3.6}$$

here, A is the matrix with rows indexed by the elements of $\mathcal{C}_0$, colums indexed by the elements of $\mathcal{D}^*$, and each entry equal to the appropriate $c^T d$.

These observations suggest an idea for finding cutters: having found a reasonably large set $\mathcal{D}^*$ of dominos, we restrict our search for cutters to subsets of $\mathcal{D}^*$. This arbitrary restriction enables us to restrict the search further, without any further loss of generality, to subsets of $\mathcal{D}^*$ defined by solutions of (3.2), (3.6): a subset of $\mathcal{D}^*$ is a cutter if and only it is defined by a solution of (3.2), (3.6) and has property (iii). To implement this idea, we need a $\mathcal{D}^*$ to start with, and that is where PQ-trees come in: any PQ-tree compatible with x that has a reasonably large number of Q-nodes is a source of a reasonably large set of dominos. Consider a partition of V into sets $V_0, V_1, \ldots, V_k$ such that, for each $i = 0, 1, \ldots, k$, setting $A = V_i$ and (with subscript arithmetic modulo $k+1$) $B = V_{i+1}$ yields a domino; we refer to this set of $k+1$ dominos as a *necklace* and to each V_i as a *bead* of this necklace. In any PQ-tree compatible with x, each Q-node v with children $v_1, v_2, \ldots, v_k$ defines a necklace by $V_0 = V - D(v)$ and $V_i = D(v_i)$; having constructed a PQ-tree compatible with x, we let $\mathcal{D}^*$ consist of all the dominos in the necklaces defined by Q-nodes of this PQ-tree.

Unfortunately, system (3.2), (3.6) may have an overwhelming number of solutions and only a small fraction of this number may have property (iii). Fortunately, the set of solutions of (3.2), (3.6) partitions naturally into sizable classes labeled in such a way that

(a) the labels of these classes can be enumerated easily one by one, and

(b) it is easy to recognize labels of classes that include at least one solution with property (iii).

As for (a), the key observation is that

$c^T d' \equiv c^T d'' \bmod 2$ whenever $c \in \mathcal{C}$ and
d', d'' are incidence vectors of two dominos that belong to the same necklace

(to see this, note that graph arising from G by shrinking each of $V_0, V_1, \ldots, V_k$ is a cycle). It follows that the set of columns of A partitions into equivalence classes of identical columns, these equivalence classes being in a one-to-one correspondence with all the Q-nodes of our PQ-tree. Let $\mathcal{Q}$ denote the set of all the Q-nodes in our PQ-tree and let B be the matrix, with rows indexed by the elements of $\mathcal{C}_0$ and colums indexed by the elements of $\mathcal{Q}$, that arises from A by replacing each equivalence class of columns with a single representative. Solutions of the system

$$Bt \equiv 0 \bmod 2, \quad e^T t \equiv 1 \bmod 2 \tag{3.7}$$

(the vector e here, having length $|\mathcal{Q}|$, is different from its counterpart in (3.2) !) are intimately related to solutions of (3.2), (3.6): with $N(w)$ standing for the necklace of w, the link is the system

$$t_w \equiv \sum_{d \in N(w)} s_d \bmod 2. \tag{3.8}$$

If s solves (3.2), (3.6) then the t defined by (3.8) solves (3.7); conversely, if t solves (3.7) then any s satisfying (3.8) solves (3.2), (3.6). In this sense, each solution of (3.7) is a label of a class of solutions of (3.2), (3.6).

As for (b), the class of vectors labeled by t includes at least one s with property (iii) if and only if the family of necklaces defined by t admits a system of pairwise vertex-disjoint representatives; in that case, we shall call the family *representable.* A fast way of finding a system of pairwise vertex-disjoint representatives (or establishing its nonexistence) relies on an easy observation, which we set apart as a lemma only for the sake of clarity.

LEMMA 3.1 *If $U_0, U_1, \ldots, U_r$ are the beads of the necklace defined by a Q-node u and $V_0, V_1, \ldots, V_s$ are the beads of the necklace defined by another Q-node v, then there are unique subscripts i and j such that $U_i \cup V_j = V$.*

PROOF. If neither u nor v is an ancestor of the other then $(V - D(u)) \cup (V - D(v)) = V$. If u is an ancestor of v then some child u_i of u is an ancestor of v (possibly $u_i = v$) and $D(u_i) \cup (V - D(v)) = V$. This establishes the existence of i and j. Their uniqueness follows easily from the fact that each of the two necklaces has at least three beads; here, no references to the PQ-tree are necessary. $\Box$

If the two necklaces of Lemma 3.1 belong to a representable family then the domino representing the first necklace must avoid U_i (otherwise it would bump into every domino of the second necklace) and the domino representing the second necklace must avoid V_j (otherwise it would bump into every domino of the first necklace); conversely, as soon as the first representative avoids U_i and the second representative avoids V_j, the two representatives are guaranteed to be vertex-disjoint.

This observation suggests an idea of a polynomial-time algorithm for recognizing representable families of necklaces defined by Q-nodes of our tree. In each iteration, we consider two of the input necklaces; to guarantee that the two dominos representing these two necklaces will be vertex-disjoint, we mark a uniquely defined bead in each of the two necklaces as taboo. If, after the last iteration, some necklace contains no two consecutive unmarked beads then we know that the family admits no system of vertex-disjoint representatives; else we may represent each necklace by a domino consisting of two consecutive unmarked beads.

To develop an implementation of this idea that runs in time $O(n)$, consider a Q-node v with children $v_1, v_2, \ldots, v_k$; in the necklace defined by v, associate each bead $D(v_i)$ with the corresponding child v_i and refer to the remaining bead $V - D(v)$ as the *outer* bead. In these terms, our taboo rules can be restated as follows:

- a child of a node v with $t_v = 1$ is taboo if and only if
  at least one node u with $t_u = 1$ is a descendant of this child,
- the outer bead of a node v with $t_v = 1$ is taboo if and only if
  at least one node u with $t_u = 1$ is not a descendant of v.

(Here, as usual, every node is considered to be its own descendant.) This formulation leads us to scan all the nodes of our PQ-tree recursively so that scanning each node is preceded by scanning all its children; while we are scanning a node v, we mark it either as `NONEMPTY` in case at least one of its children is marked `NONEMPTY` and/or v is a Q-node with $t_v = 1$; in all the remaining cases, we mark v as `EMPTY`. In addition, if v is a Q-node with $t_v = 1$, then we also assign a representative domino to its necklace or else return a message indicating that the family is not representable. If v has two consecutive children marked `EMPTY` then the necklace may be represented by the domino defined by these two children. If no two consecutive children of v are marked EMPTY then the family is not representable unless all the nodes u with $t_u = 1$ are descendants of v; in this exceptional case, we may also have the option of representing the necklace by the domino defined by an end-child (that is, the very first child or the very last child) of v along with the outer bead; this option is available if and only if the end-child is marked EMPTY. To find out in constant time whether we are dealing with the exceptional case or not, we may keep a pointer to the node α that, among all the nodes w with $t_w = 1$, comes first in the preorder and a counter of the number of necklaces represented so far: we are dealing with the exceptional case if and only if $v = \alpha$ and the counter is at the total number of necklaces minus one.

Now we shall illustrate the method of this section in its entirety on the x of Figure 1.1 and the PQ-tree of Figure 2.1 with the fundamental set of cycles indicated in Figure 3.2 (the spanning tree has solid edges; the dashed edges define the cycles A,B, . . . , I in the fundamental set).

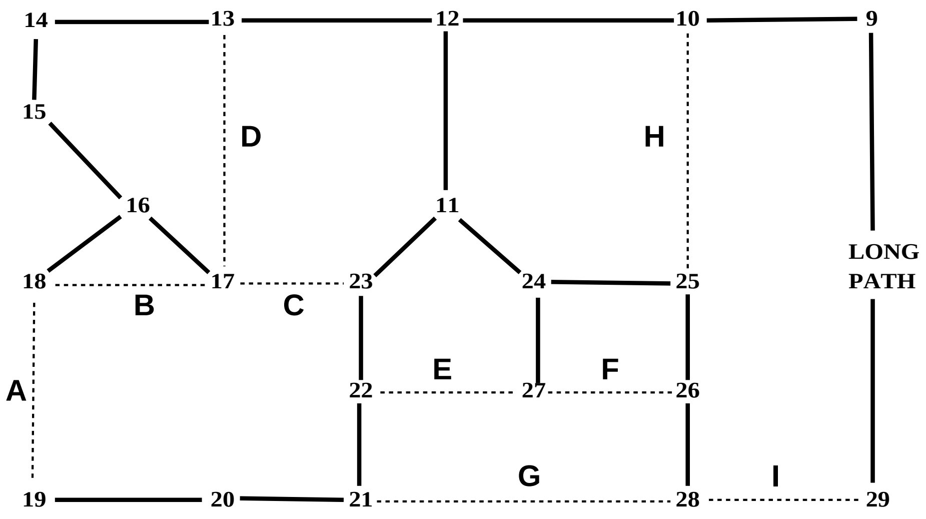

Fig.3.2: A spanning tree and its fundamental set of cycles

In this case, system (3.7) assumes the form

$$\begin{aligned}
t_{44}+t_{45}+t_{46}+t_{47}+t_{52}+t_{55} &\equiv 0 \bmod 2\\
t_{50} &\equiv 0 \bmod 2\\
t_{44}+t_{45}+t_{50}+t_{55} &\equiv 0 \bmod 2\\
t_{45} &\equiv 0 \bmod 2\\
t_{47}+t_{51}+t_{54} &\equiv 0 \bmod 2\\
t_{48}+t_{49} &\equiv 0 \bmod 2\\
t_{47}+t_{48}+t_{51}+t_{53}+t_{54}+t_{55} &\equiv 0 \bmod 2\\
t_{44}+t_{48}+t_{53} &\equiv 0 \bmod 2\\
t_{43}+t_{44}+t_{48}+t_{51}+t_{53} &\equiv 0 \bmod 2\\
t_{43}+t_{44}+t_{45}+t_{46}+t_{47}+t_{48}+t_{49}+t_{50}+t_{51}+t_{52}+t_{53}+t_{54}+t_{55} &\equiv 1 \bmod 2.
\end{aligned}$$

Its solution space has dimension four: a general solution can be obtained by first choosing $t_{44}, t_{46}, t_{47}, t_{48}$ arbitrarily, and then setting

$$\begin{aligned}
t_{43} &\equiv t_{44}+t_{47}+t_{48}+1 \bmod 2,\\
t_{45} &\equiv 0 \bmod 2,\\
t_{49} &\equiv t_{48} \bmod 2,\\
t_{50} &\equiv 0 \bmod 2,\\
t_{51} &\equiv t_{44}+t_{47}+t_{48}+1 \bmod 2,\\
t_{52} &\equiv t_{46}+t_{47} \bmod 2,\\
t_{53} &\equiv t_{44}+t_{48} \bmod 2,\\
t_{54} &\equiv t_{44}+t_{48}+1 \bmod 2,\\
t_{55} &\equiv t_{44} \bmod 2.
\end{aligned}$$

Of the resulting sixteen solutions, five are representable:

- $t_{43}=t_{51}=t_{54}=1$ (with $t_w=0$ for all remaining w) yields, for instance, the comb
  $H=\{10,11,12,24,25\}$,
  $T_1=\{9,10\}$,
  $T_2=\{24,25,26,27\}$,
  $T_3=\{11,12,\ldots,23\}$.

- $t_{44}=t_{53}=t_{55}=1$ (with $t_w=0$ for all remaining w) yields the comb
  $H=\{11,22,23,\ldots,27\}$,
  $T_1=\{11,12\}$,
  $T_2=\{24,25,\ldots,42,1,2,\ldots,10\}$,
  $T_3=\{13,14,\ldots,23\}$.

- $t_{46} = t_{47} = t_{54} = 1$ (with $t_w = 0$ for all remaining w) yields, for instance, the comb
  $H = \{11, 12, \dots, 17, 18, 23\}$,
  $T_1 = \{18, 19\}$,
  $T_2 = \{22, 23\}$,
  $T_3 = \{24, 25, \dots, 42, 1, 2, \dots, 12\}$.
- $t_{47} = t_{52} = t_{54} = 1$ (with $t_w = 0$ for all remaining w) yields the comb
  $H = \{11, 12, \dots, 16, 17, 23\}$,
  $T_1 = \{22, 23\}$,
  $T_2 = \{13, 14, \dots, 21\}$,
  $T_3 = \{24, 25, \dots, 42, 1, 2, \dots, 12\}$.
- $t_{48} = t_{49} = t_{53} = 1$ (with $t_w = 0$ for all remaining w) yields the comb
  $H = \{25, 26, 28, 29, 30 \dots, 42, 1, 2, \dots, 10\}$,
  $T_1 = \{24, 25\}$,
  $T_2 = \{26, 27\}$,
  $T_3 = \{28, 29, \dots, 42, 1, 2, \dots, 23\}$.

(These five combs are depicted in Figures 3.3–3.7.)

Incidentally, this example illustrates the fact that our method yields more cuts than we have initially admitted. Each necklace with beads $V_0, V_1, \dots, V_d$ such that $d \geq 3$ yields more than $d+1$ dominos: for each choice of distinct i, j, k, in this cyclic order,

$$A = V_{i+1} \cup V_{i+2} \cup \dots \cup V_j \text{ and } B = V_{j+1} \cup V_{j+2} \cup \dots \cup V_k$$

yield a domino. Since the edge-set of this new domino equals the edge-set of the the domino with $A = V_j$, $B = V_{j+1}$, the new domino may also be used to represent the necklace. In the example, $t_{43} = t_{51} = t_{54} = 1$ yields 2600 distinct comb inequalities violated by x (there are 2600 choices of T_1) and $t_{46} = t_{47} = t_{54} = 1$ yields 10 distinct comb inequalities violated by x (there are 10 choices of T_1).

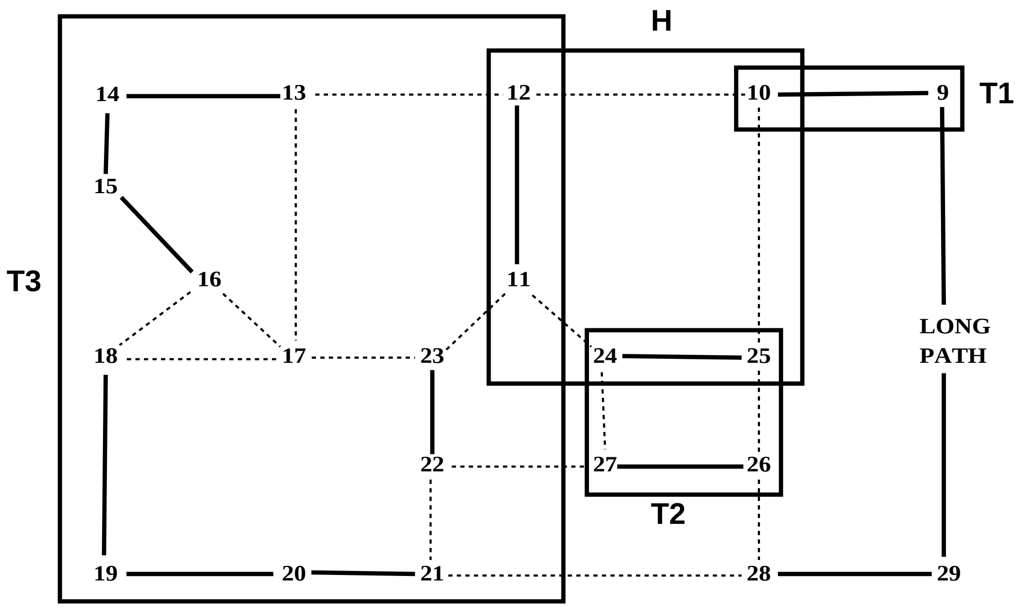

Figure 3.3: First comb for the forty-two cities ($t_{43} = t_{51} = t_{54} = 1$).

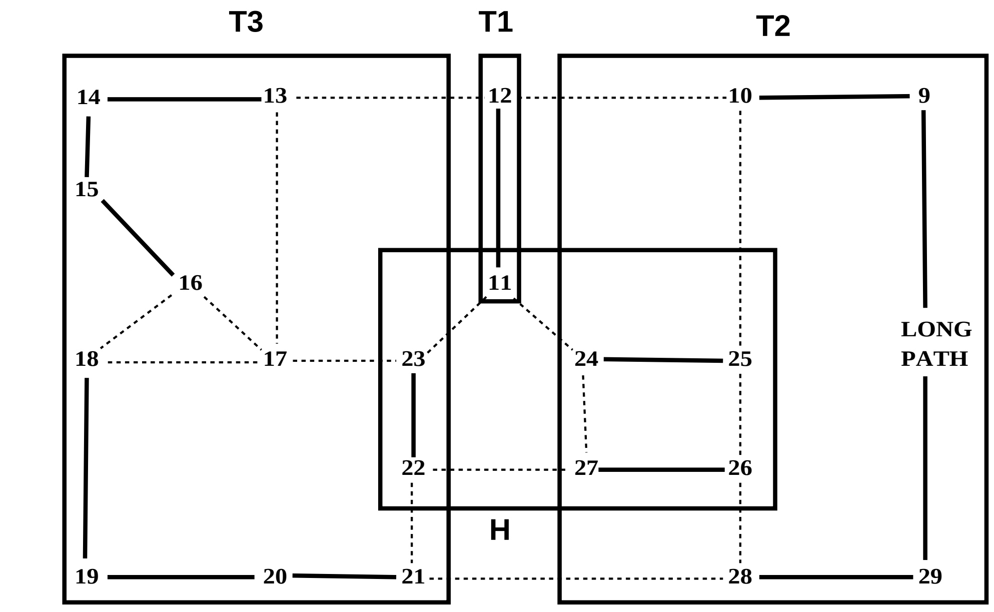

Figure 3.4: Second comb for the forty-two cities ($t_{44} = t_{53} = t_{55} = 1$).

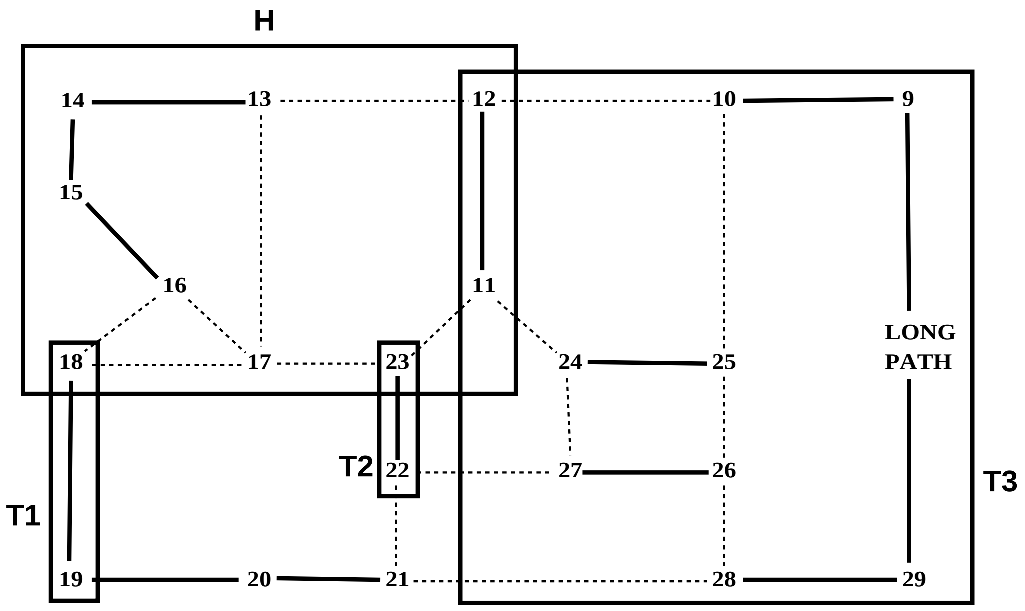

Figure 3.5: Third comb for the forty-two cities ($t_{46} = t_{47} = t_{54} = 1$).

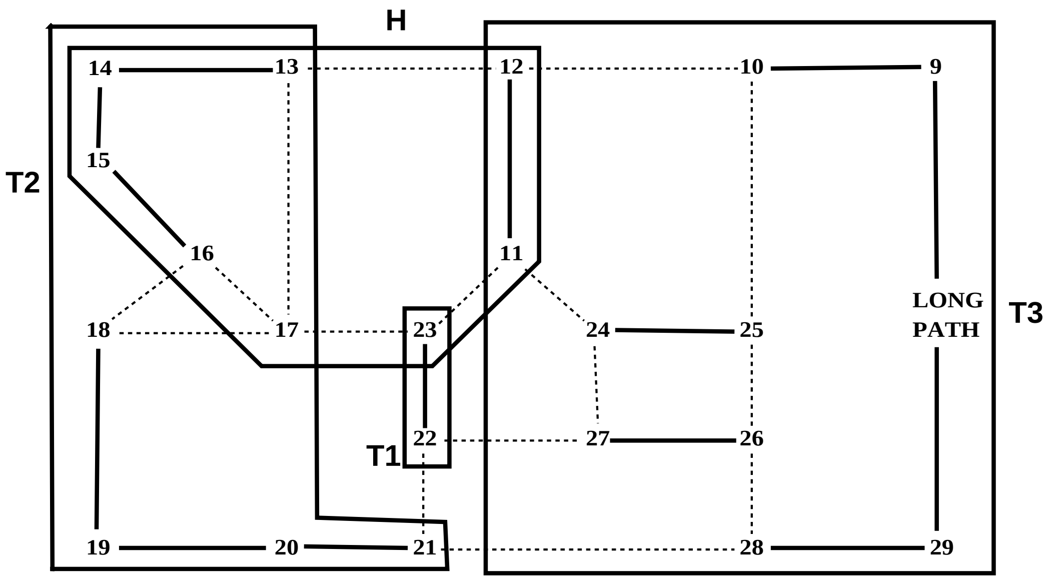

Figure 3.6: Fourth comb for the forty-two cities ($t_{47} = t_{52} = t_{54} = 1$).

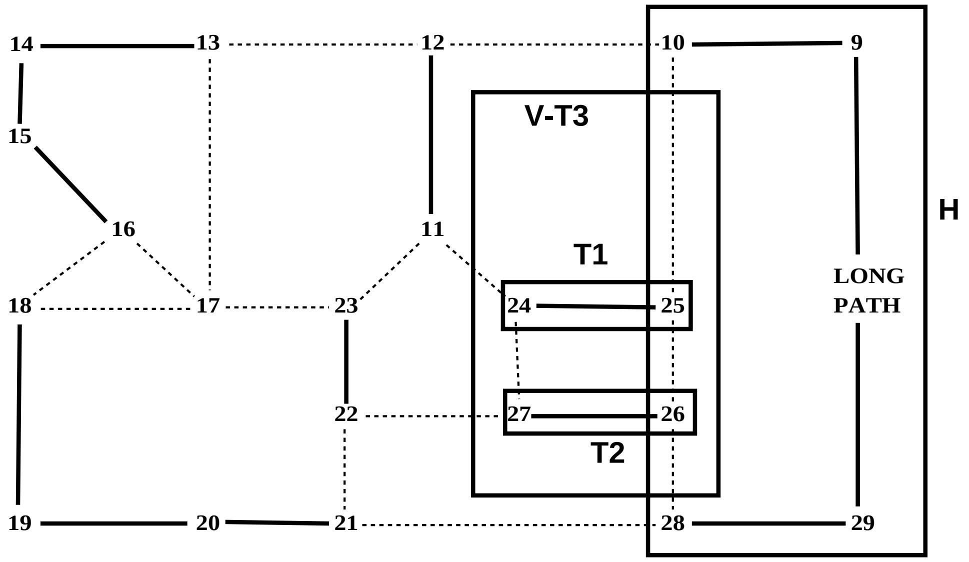

Figure 3.7: Fifth comb for the forty-two cities ($t_{48} = t_{49} = t_{53} = 1$).

In general, the solution space of system (3.7) may have a large dimension, which makes enumerating *all* solutions out of the question. If that is the case then, having transformed (3.7) into an echelon form, we may generate its solutions one by one and test each of them for representability until our patience runs out. Here, the chances of coming across a representable solution can be improved by a simple trick. Let us say that a zero-one vector vector t *majorizes* a zero-one vector t' of the same length if each component of t is at least the corresponding component of t' (to put it differently, t' has a 0 wherever t does). Trivially, if a representable solution t of (3.7) majorizes another solution t' then t' is also representable. In particular, if (3.7) has any representable solution at all, then it has a representable solution which is *minimal* in the sense that it majorizes no other solution of (3.7). This observation leads us to replace each new solution t that we generate with a minimal solution t' majorized by t and then test t' rather than t for representability.

In our implementation, the dimension of of the solution space of (3.7) is considered large if it is at least six; our patience runs out when we have generated a solution t of (3.7), reduced t to a minimal solution t', and tested t' for representability fifty times; both of these thresholds have been chosen arbitrarily at least to some extent. Led by the belief that random samples from the uniform distribution over all minimal solutions are better than biased samples, we use randomization in generating each t as well as in the subsequent reduction of t into a t'. To generate t, we assign zero-one values to the free variables at random and solve for the remaining variables. To reduce t to t', we remove all the variables whose current values are zero, transform the resulting system into an echelon form again if necessary, and repeat the whole process (assign zero-one values to the free variables at random, solve for the remaining variables, remove all the variables whose current values are zero, and transform the resulting system into an echelon form again if necessary) as long as any free variables remain.

The method described so far finds either comb inequalities violated by a large amount (in the sense that the left-hand side of (3.1) with our x exceeds the right-hand side by 0.5) or no violated comb inequalities at all. Its extension that we are going to describe now settles for less and may find more.

To begin, consider any set F of edges with $\delta = \sum_{e \in F} x_e < 1$ and suppose that we have found a set of vertex-disjoint dominos $D_1, D_2, \ldots, D_{2k+1}$ (along with a partition of the vertex-set of each D_i into tight sets A_i, B_i) such that the union of the edge-sets of our $2k+1$ dominos is an edge-cut in $G - F$. Now we can find a set H of vertices such that an edge of $G - F$ has one endpoint in H and other endpoint outside H if and only if it is an edge of some D_i. Note first that each D_i has at least one edge outside F (since $\delta < 1 = x(A_i, B_i)$); then, writing $T_i = A_i \cup B_i$ again, that H meets every T_i but does not contain any; and finally that $x(H, V - H) \le \delta + \sum x(A_i, B_i) = 2k + 1 + \delta$, which implies $x(H) \ge (|H| - (1 + \delta)/2) - k$ by virtue of (2.2). Hence (3.1) is a comb inequality violated by x. This observation suggests applying the method not just to G, but also to graphs arising from G by removing sets F of edges e with small x_e: possibly, comb inequalities violated by x can be obtained from vectors that satisfy systems (3.7) arising from $G - F$ but do not satisfy the system (3.7) arising from G.

Our implementation of this idea goes as follows. With the *weight* of each edge e defined as x_e, we construct a maximum-weight spanning tree T of G and define (3.7) by the fundamental set of cycles with respect to this T. Except for $e^T t \equiv 1 \bmod 2$, each congruence in (3.7) arises from some cycle in the fundamental set, and therefore from some edge e of $G - T$; let us say that the *weight* of this congruence is the corresponding x_e. We construct a sequence of subsystems of (3.7) by starting with $e^T t \equiv 1 \bmod 2$ and then bringing in the remaining congruences one by one in a nonincreasing order of their weights. With each new arrival, the current system is transformed into an echelon form; if the newcomer made the system unsolvable then it is deleted at once. We find the appropriate number of minimal solutions, test them for representability, and see if their representations yield comb inequalities violated by our x not only when all the congruences from (3.7) have been considered, but possibly also earlier. More precisely, we monitor the number m of free variables in our current system and compare it to some benchmark initialized as the number of variables in (3.7). Whenever m drops below two thirds of the benchmark, we find the appropriate number of minimal solutions, test them for representability, and see if their representations yield comb inequalities violated by our x; then we reset the benchmark to m.

In closing, let us remark that a system (3.7) whose solution space has a large dimension may look like good news: the more solutions there are altogether, the better seem the chances that there is a representable solution among them. However, the solution space of (3.7) may have an unnaturally large dimension simply because (3.7) has duplicated columns. In our example, Q-nodes 46 and 52 give rise to identical columns in (3.7); the two necklaces defined by these two nodes can be simultaneously refined into a single necklace with beads

$$\{22, 23, \ldots, 42, 1, 2, \ldots, 12\}, \{13, 14, 15, 16, 17\}, \{18\}, \{19\}, \{20\}, \{21\};$$

in the PQ-tree shown in Figure 3.8, compatible with x, this necklace is defined by the single Q-node 52. In terms of (3.7), replacing the PQ-tree of Figure 2.1 with the PQ-tree of Figure 3.8 brings about a removal of variable t_{46} ; as a result, the dimension of the solution space drops by one, the number of solutions is halved, and yet no representable solutions are lost. In fact, presence of duplicated columns in (3.7) *always* indicates either the existence of a PQ-tree whose necklaces refine the necklaces of the original tree or the presence of a bad set: this is an immediate consequence of the following theorem combined with Theorem 4.1 of the next section.

THEOREM 3.1 *There is a polynomial time algorithm that, given any PQ-tree T compatible with x such that the resulting system (3.7) has duplicated columns, returns a tight subset S of W that is not in $\mathcal{B}(T)$.*

PROOF. Let u and v be the two Q-nodes that give rise to identical columns in (3.7). It is a trivial consequence of Lemma 3.1 that there are vertex-disjoint dominos D_u and D_v such that D_u belongs to the necklace defined by u and D_v belongs to the necklace defined by v. If d_u and d_v denote the incidence vectors of the edge-sets of these two dominos then, since u and v define identical columns in (3.7), we have $c^T d_u \equiv c^T d_v \bmod 2$ for all c in $\mathcal{C}_0$, and so $c^T(d_u + d_v) \equiv 0 \bmod 2$ for all c in $\mathcal{C}_0$; since D_u and D_v are disjoint, this means that the union of their edge-sets is an edge-cut. Partition V into nonempty sets S and $V - S$ with $S \subseteq W$ so that an edge of G has one endpoint in S and the other endpoint in $V - S$ if and only if it is an edge of D_u or D_v; return S.

To see that S is tight, observe that $x(S, V-S) = x^T(d_u + d_v) = 2$ since $x^T d_u = x^T d_v = 1$. To see that $S \notin \mathcal{B}(T)$, swap first A_u and B_u if necessary and then A_v and B_v if necessary to obtain

$$S \cap A_u \neq \emptyset, \quad S \not\supseteq B_u, \quad S \cap A_v \neq \emptyset, \quad S \not\supseteq B_v. \tag{3.9}$$

It is a routine matter to verify that no member of $\mathcal{B}(T)$ satisfies (3.9) in place of S. □

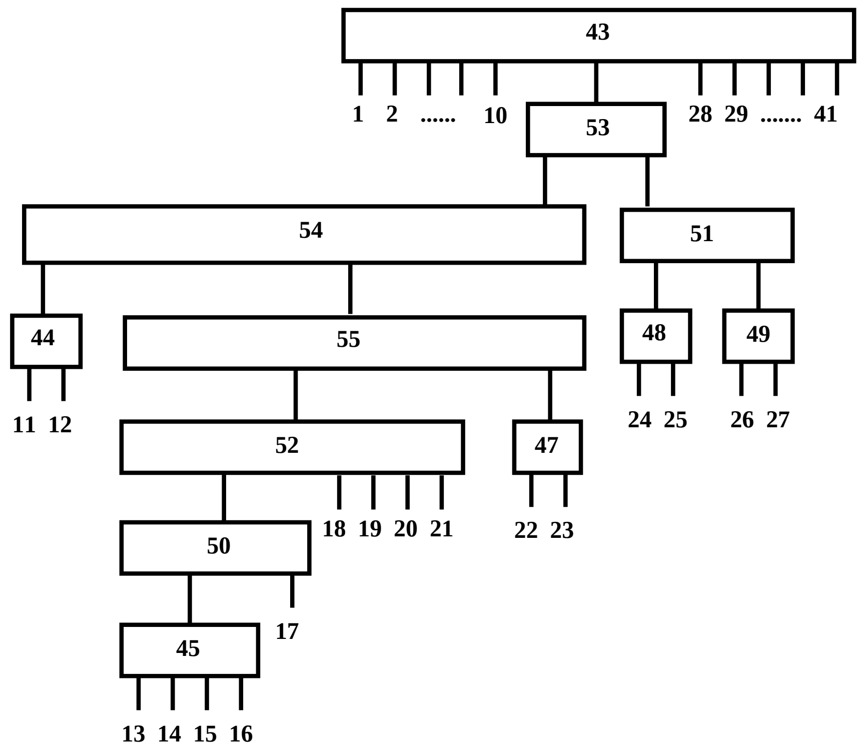

Figure 3.8: A better PQ-tree compatible with the vector in Fig.1.1.

4 REFINING PQ-TREES

Every PQ-tree compatible with x can be thought of as a hieararchical decomposition of V with respect to x. Or it can be thought of as just a compact device that stores the family $\mathcal{B}(T)$ of tight sets. Given a PQ-tree T compatible with x and given a tight subset S of W ($= V-$`exterior`) that is not in $\mathcal{B}(T)$, can we construct a PQ-tree T' compatible with x such that $\mathcal{B}(T') \supseteq \mathcal{B}(T) \cup \{S\}$? The following answer is adequate for our purpose.

THEOREM 4.1 *There is a polynomial-time algorithm that, given any PQ-tree T compatible with x, and any tight subset S of W that is not in $\mathcal{B}(T)$, returns either a PQ-tree T' compatible with x such that $\mathcal{B}(T') \supseteq \mathcal{B}(T) \cup \{S\}$ or a bad set.* □

The algorithm of Theorem 4.1 is the subject of the present section.

Throughout this section, S will denote an arbitrary but fixed tight subset of W. A node u of a PQ-tree whose set of leaves is W will be referred to as `FULL` if $D(u)$ is contained in S, as `EMPTY` if $D(u)$ is disjoint from S, and as as `PARTIAL` otherwise. A node will be called `SPECIAL` if and only if it is a `PARTIAL` Q-node with no `PARTIAL` children and such that either all its `EMPTY` children precede all its `FULL` children or the other way around.

LEMMA 4.1 *Every node of a PQ-tree compatible with x has at most two* `SPECIAL` *children.*

PROOF. Assuming the contrary, consider a node with `SPECIAL` children v_1, v_2, v_3. Since $D(v_1)$, $D(v_2)$, $D(v_3)$ are pairwise disjoint, we have

$$x(S, V-S) \geq \sum_{i=1}^{3} x(D(v_i) \cap S, D(v_i) - S) = 3,$$

and so S is not tight, a contradiction. □

LEMMA 4.2 *If a node u of a PQ-tree compatible with x has two* `SPECIAL` *children and $D(u) \not\supseteq S$ then at least one of $S - D(u)$ and $S \cup D(u)$ is bad.*

PROOF. Let v_1, v_2 be the two `SPECIAL` children of u. Writing $A = D(u) - S$, $B = D(u) \cap S$, $C = S - D(u)$ and $D = V - (A \cup B \cup C)$, we have

$$x(A, B) \geq \sum_{i=1}^{2} x(D(v_i) - S, D(v_i) \cap S) = 2\ ;$$

since S is tight, it follows that

$$x(B, D) = x(C, A) = x(C, D) = 0\ .$$

Now $x(C, V-C) + x(V-D, D) = x(C, B) + x(A, D) = x(D(u), V - D(u)) = 2$ and the desired conclusion follows. □

Let T be a PQ-tree compatible with x, let u be a Q-node with children $u_1, u_2, \ldots, u_r$ (in the left-to-right order) and let u_i be a Q-node with children $v_1, v_2, \ldots, v_s$ (in the left-to-right order). By *splicing* u_i, we shall mean first substituting the sequence $v_1, v_2, \ldots, v_s$ for u_i on the list of children of u and then removing u_i from the tree altogether.

LEMMA 4.3 *Let T be a PQ-tree compatible with x, let u be a Q-node with children $u_1, u_2, \ldots, u_r$ (in the left-to-right order) and let u_i be a* `SPECIAL` *node with children $v_1, v_2, \ldots, v_s$ (in the left-to-right order). If*

- *v_1 is* `FULL` *and u_{i-1} is* `FULL`*, or*
- *v_s is* `FULL` *and u_{i+1} is* `FULL`

then the tree obtained from T by splicing u_i is compatible with x or else $S \cup D(u_i)$ is bad.

PROOF. Symmetry allows us to restrict ourselves to the case where v_1 is `FULL` and u_{i-1} is `FULL`. Writing

$$A = D(u_i) \cap S \ , \quad B = D(u_i) - S \ ,$$

note that B is tight and that $x(A, B) = 1$. If $x(D(u_{i-1}), B) > 0$ then $S \cup D(u_i)$ is bad; now we may assume that $x(D(u_{i-1}), B) = 0$. Since $x(D(u_{i-1}), D(u_i)) = 1$ and since $D(v_1), D(v_2), \ldots, D(v_s)$ are tight sets with $x(D(v_j), D(v_{j+1})) = 1$ whenever $1 \leq j < s$, it follows that $x(D(u_{i-1}), D(v_1)) = 1$; if $i < r$ then $x(D(u_i), D(u_{i+1})) = 1$ allows us to conclude also that $x(D(v_s), D(u_{i+1})) = 1$. □

LEMMA 4.4 *Let T be a PQ-tree compatible with x, let u be a Q-node with children $u_1, u_2, \ldots, u_r$ (in the left-to-right order) and let u_i be a* `SPECIAL` *node with children $v_1, v_2, \ldots, v_s$ (in the left-to-right order). If*

- *v_1 is* `EMPTY` *and u_{i-1} is* `EMPTY`*, or*
- *v_s is* `EMPTY` *and u_{i+1} is* `EMPTY`

then the tree obtained from T by splicing u_i is compatible with x or else $S - D(u_i)$ is bad.

PROOF. Symmetry allows us to restrict ourselves to the case where v_1 is `EMPTY` and u_{i-1} is `EMPTY`. Now we may follow the proof of Lemma 4.3 with a single modification: if $x(D(u_{i-1}), B) > 0$ then $S - D(u_i)$ is bad. □

LEMMA 4.5 *Let T be a PQ-tree compatible with x, let u be a Q-node with children $u_1, u_2, \ldots, u_d$ (in the left-to-right order), let u_i be a* `SPECIAL` *node with children $v_1, v_2, \ldots, v_s$ (in the left-to-right order), let u_{i+1} be a* `SPECIAL` *node with children $w_1, w_2, \ldots, w_s$ (in the left-to-right order). If*

- *v_r is* `FULL` *and w_1 is* `FULL`

then the tree obtained from T by splicing u_i and u_{i+1} is compatible with x or else $S \cup D(u_i) \cup D(u_{i+1})$ is bad.

PROOF. Writing

$$A_1 = D(u_i) \cap S \ , \quad B_1 = D(u_i) - S \ , \quad A_2 = D(u_{i+1}) \cap S \ , \quad B_2 = D(u_{i+1}) - S \ ,$$

note that A_1, B_1, A_2, B_2 are tight and that $x(A_1, B_1) = 1$, $x(A_2, B_2) = 1$. If $x(A_1, B_2) + x(B_1, B_2) + x(B_1, A_2) > 0$ then $S \cup D(u_i) \cup D(u_{i+1})$ is bad; else $x(D(u_i), D(u_{i+1})) = 1$ implies that $x(A_1, A_2) = 1$. Since $D(v_1), D(v_2), \ldots, D(v_r)$ are tight sets with $x(D(v_j), D(v_{j+1})) = 1$ whenever $1 \leq j < r$, and since $D(w_1), D(w_2), \ldots, D(w_s)$ are tight sets with $x(D(w_k), D(w_{k+1})) = 1$ whenever $1 \leq k < s$, it follows that $x(D(v_r), D(w_1)) = 1$; if $i+1 < d$ then $x(D(u_{i+1}), D(u_{i+2})) = 1$ allows us to conclude also that $x(D(w_s), D(u_{i+2})) = 1$; if $i > 1$ then $x(D(u_{i-1}), D(u_i)) = 1$ allows us to conclude also that $x(D(u_{i-1}), D(v_1)) = 1$. □

LEMMA 4.6 *Let T be a PQ-tree compatible with x and let u be a Q-node whose end-children are both* `EMPTY`. *If $S \cap D(u) \in \mathcal{B}(T)$ and $S \not\subseteq D(u)$ then $S - D(u)$ is bad.*

PROOF. Let $u_1, u_2, \ldots, u_d$ be the children of u (in the left-to-right order). Since $D(u_1), D(u_2), \ldots, D(u_d)$ are tight sets with $x(D(u_j), D(u_{j+1})) = 1$ for all $j = 1, 2, \ldots, d-1$, we have

$$x(D(u_2) \cup D(u_3) \cup \ldots \cup D(u_{d-1}), V - D(u)) = 0;$$

in particular, $x(S \cap D(u), S - D(u)) = 0$. The desired conclusion follows since S and $S \cap D(u)$ are tight and $S - D(u)$ is nonempty. □

LEMMA 4.7 *Let V be partitioned into pairwise disjoint nonempty sets A, B, C, D. If $A \cup B$ and $B \cup C$ are both tight and if none of A, B, C, D are bad then all of A, B, C, D are tight.*

PROOF. By assumption, we have

$$x(A \cup B, C \cup D) = 2,$$

$$x(B \cup C, A \cup D) = 2,$$

and

$$1 \leq \frac{1}{2} x(A, B \cup C \cup D),$$

$$1 \leq \frac{1}{2} x(B, A \cup C \cup D),$$

$$1 \leq \frac{1}{2} x(C, A \cup B \cup D),$$

$$1 \leq \frac{1}{2} x(D, A \cup B \cup C);$$

since x satisfies (2.1), we have

$$-x(A, C) \leq 0,$$

$$-x(B, D) \leq 0.$$

The sum of these eight relations reads $0 \leq 0$, and so all of them must hold as equations. □

For any node u of a PQ-tree compatible with x, we set

$$\begin{aligned} F(u) &= \cup\{D(v) : v \text{ is a } \texttt{FULL} \text{ child of } u\}, \\ E(u) &= \cup\{D(v) : v \text{ is an } \texttt{EMPTY} \text{ child of } u\}. \end{aligned}$$

LEMMA 4.8 *There is a polynomial time algorithm that, given any PQ-tree T compatible with x and any node u of T such that $F(u)$ is neither tight nor empty, returns a bad set.*

PROOF. Repeat for all `PARTIAL` children v of u: If one of the four sets $S - D(v)$, $S \cap D(v)$, $D(v) - S$, $S \cup D(v)$ is bad then return this bad set; else replace S with $S - D(v)$ (which is guaranteed by Lemma 4.7 to remain tight).

Now $S \cap D(u) = F(u)$. By assumption, $F(u)$ is neither tight nor empty; hence Lemma 4.7 guarantees that one of the four sets $S - D(u)$, $S \cap D(u)$, $D(u) - S$, $S \cup D(u)$ must be bad; return this bad set. □

LEMMA 4.9 *There is a polynomial time algorithm that, given any PQ-tree T compatible with x and any node u of T such that $S \not\subseteq D(u)$ and $E(u)$ is neither tight nor empty, returns a bad set.*

PROOF. Repeat for all `PARTIAL` children v of u: If one of the four sets $S - D(v)$, $S \cap D(v)$, $D(v) - S$, $S \cup D(v)$ is bad then return this bad set; else replace S with $S \cup D(v)$ (which is guaranteed by Lemma 4.7 to remain tight).

Now $D(u) - S = E(u)$. By assumption, $E(u)$ is neither tight nor empty; hence Lemma 4.7 guarantees that one of the four sets $S - D(u)$, $S \cap D(u)$, $D(u) - S$, $S \cup D(u)$ must be bad; return this bad set. □

LEMMA 4.10 *There is a polynomial time algorithm that, given any PQ-tree T compatible with x and any node u of T such that $D(u) - E(u)$ is neither tight nor empty, returns a bad set.*

PROOF. Repeat for all `PARTIAL` children v of u: If one of the four sets $S - D(v)$, $S \cap D(v)$, $D(v) - S$, $S \cup D(v)$ is bad then return this bad set; else replace S with $S \cup D(v)$ (which is guaranteed by Lemma 4.7 to remain tight).

Now $S \cap D(u) = D(u) - E(u)$. By assumption, $D(u) - E(u)$ is neither tight nor empty; hence Lemma 4.7 guarantees that one of the four sets $S - D(u)$, $S \cap D(u)$, $D(u) - S$, $S \cup D(u)$ must be bad; return this bad set. □

LEMMA 4.11 *There is a polynomial time algorithm that, given any PQ-tree T compatible with x, any Q-node u of T with children $u_1, u_2, \ldots, u_d$ (in the left-to-right order), and any subscripts i, j, k such that $i < j < k$, u_i is not* `EMPTY`*, u_j is not* `FULL`*, and u_k is not* `EMPTY`*, returns a bad set.*

PROOF. If one of the four sets $S - D(u_j)$, $S \cap D(u_j)$, $D(u_j) - S$, $S \cup D(u_j)$ is bad then return this bad set; else replace S with $S - D(u_j)$ (which is guaranteed by Lemma 4.7 to remain tight).

Writing $L = D(u_1) \cup D(u_2) \cup \ldots \cup D(u_{j-1})$ and $R = D(u_{j+1}) \cup D(u_{j+2}) \cup \ldots \cup D(u_d)$, note that $S \cap D(u) \subseteq L \cup R$, $S \cap L \neq \emptyset$, $S \cap R \neq \emptyset$, and $x(L, R) = 0$. If $S \subseteq D(u)$ then one of $S \cap L$, $S \cap R$ must be bad; else Lemma 4.7 guarantees that one of the four sets $S - D(u)$, $S \cap D(u)$, $D(u) - S$, $S \cup D(u)$ must be bad. Return the bad set. □

PROOF OF THEOREM 4.1. Let `top` denote the lowest node u in T such that $D(u)$ contains S. Scan all the `PARTIAL` descendants of `top`, in any order such that all the `PARTIAL` children of a node are scanned before the node itself, and process each node while scanning it. Processing u means replacing T (if necessary) with a PQ-tree T' such that

- T' is compatible with x,
- $\mathcal{B}(T) \subseteq \mathcal{B}(T')$,
- T and T' differ (if at all) only in their subtrees rooted at u,
- $S \cap D(u) \in \mathcal{B}(T')$,
- unless u is `top`, it is `SPECIAL` in T';

if T cannot be restructured in this way then a bad set is discovered immediately. Note that, when u comes up for processing, all of its `PARTIAL` children are `SPECIAL`; in particular, Lemma 4.1 guarantees that u has at most two `PARTIAL` children now. The details of processing go as follows.

Case Q: u is a Q-node with children $u_1, u_2, \ldots, u_d$ in the left-to-right order.

Find the smallest subscript i such that u_i is not `EMPTY` and find the largest subscript k such that u_k is not `EMPTY`.

If some u_j with $i < j < k$ is not `FULL` then use Lemma 4.11 to return a bad set.

If $i = k$ then replace i with d and replace k with 1. (The only purpose of this bizarre instruction is to avoid treating the special case $i = k$ separately.)

If u_i is `PARTIAL` then reverse the list of its children if necessary so that the last child is `FULL` and then splice u_i. If u_k is `PARTIAL` then reverse the list of its children if necessary so that the first child is `FULL` and then splice u_k. If the new T is not compatible with x then use one of Lemmas 4.3, 4.4, 4.5 as appropriate to return a bad set.

If u is not `top` and its end-children (in the new T) are both `EMPTY` then use Lemma 4.6 to return a bad set.

Case P: u is a P-node.

If u has at least two `FULL` children then introduce a new child v of u and make v the new parent of all the `FULL` children of u; the new node v is a Q-node if it has precisely two children and a P-node otherwise. If the new T is not compatible with x then use Lemma 4.8 to return a bad set.

Subcase P1: $u \neq$ `top`.

If u has two `PARTIAL` children then use Lemma 4.2 to return a bad set.

If u has at least two `EMPTY` children then introduce a new child w of u and make w the new parent of all the `EMPTY` children of u; the new node w is a Q-node if it has precisely two children and a P-node otherwise. If the new T is not compatible with x then use Lemma 4.9 to return a bad set.

(Now u has at most one `FULL` child, at most one `EMPTY` child, at at most one `PARTIAL` child.)

If u has only two children then make it a Q-node and process it as a Q-node; else give the `FULL` and the `PARTIAL` child of u a new parent y and process y as a Q-node, then make y a child of u (whose other child is the remaining `EMPTY` child) and process u as a Q-node.

Subcase P2: $u =$ `top`.

(Now at least two children of u are not `EMPTY`, at most two are `PARTIAL`, and at most one is `FULL`.)

If u has at least one `EMPTY` child then introduce a new child w of u, make w the new parent of all the children of u that are not `EMPTY`, make w a P-node, and set `top` $= w$. If the new T is not compatible with x then use Lemma 4.10 to return a bad set.

Reorder the children of `top` if necessary so that the only `FULL` child (if any) comes second; then make `top` a Q-node and process it as a Q-node. □

Readers familiar with [2] will recognize in our proof of Theorem 4.1 a variation on an algorithm designed by Booth and Lueker to solve the following problem: Given a PQ-tree T and a set S of leaves of T such that $S \notin \mathcal{B}(T)$, either construct the coarsest PQ-tree T' such that

$$\mathcal{B}(T') \supseteq \mathcal{B}(T) \cup \{S\} \tag{4.1}$$

or return a failure mesage to indicate that no PQ-tree T' satisfies (4.1). Here, a PQ-tree T_1 is considered coarser than a PQ-tree T_2 if $\mathcal{B}(T_1) \subset \mathcal{B}(T_2)$; it may not be obvious that the set of all T' with property (4.1) includes the coarsest element as long as it is nonempty. The original Booth-Lueker theme can be reconstructed from our variation by simple modifications:

(i) Delete all instructions of the type "If the new T is not compatible with x then … ".

(ii) Replace all the remaining instructions of the type "use Lemma X to return a bad set" with "return a failure message".

(iii) Begin processing each u with the new instruction
"If u has at least three `PARTIAL` children then return a failure message."

If the Booth-Lueker algorithm returns a failure message then it may have modified its input tree T beyond recovery. To maintain access to T even in this case, we could make a spare copy of T before applying the algorithm; alternatively, we may first call yet another variation on the Booth-Lueker theme, one which leaves T intact and simply answers the question whether or not some PQ-tree T' satisfies (4.1). In this variation, we again scan all the `PARTIAL` descendants of `top`, in any order such that all the `PARTIAL` children of a node are scanned before the node itself, and process each node while scanning it. But processing u means something simpler now: its details go as follows.

If u has at least three `PARTIAL` children then return `NO`.
If u is not `top` and has two `PARTIAL` children then return `NO`.

Case Q: u is a Q-node with children $u_1, u_2, \ldots, u_d$ in the left-to-right order.
Find the smallest subscript i such that u_i is not `EMPTY` and find the largest subscript k such that u_k is not `EMPTY`.
If some u_j with $i < j < k$ is not `FULL` then return `NO`.
If $u =$ `top` then return `YES`.
If neither of u_1 and u_d is `FULL` and at least one of $u_2, u_3, \ldots, u_{d-1}$ is not `EMPTY` then return `NO`.

Case P: u is a P-node.
If $u =$ `top` then return `YES`.

5 THE CUTPOOL AND HOW WE STORE IT

The problem of finding cuts may seem to be this: Given an x that is not a convex combination of tours, find a hyperplane separating x from all the tours. But this formulation makes the problem sound harder than it is: we are not given x alone. We also have the current LP relaxation at our fingertips; in addition, nothing (except memory limitations) prevents us from storing at least some of the constraints added to the various LP relaxations in the past. In the subsequent three sections, we shall describe three different ways of exploiting this additional information; the purpose of the present section is only to specify just what this additional information is and how we store it.

Let us begin with a few definitions.

Given any vector x satisfying (2.1), (2.2), we define the *deficiency* of each subset S of V as

$$\delta_x(S) = x(S, V-S) - 2;$$

whenever x is specified by the context, we allow ourselves to write simply $\delta(S)$ rather than $\delta_x(S)$.

A *hypergraph* is an ordered pair $(V, \mathcal{F})$ such that V is a finite set and $\mathcal{F}$ is a family of (not necessarily distinct) subsets of V; elements of V are called the *vertices* of the hypergraph and the elements of $\mathcal{F}$ are called the *edges* of the hypergraph. Given a hypergraph $(V, \mathcal{F})$ denoted $\mathcal{H}$, and given a vector x, whose components are indexed by the edges of the complete graph with vertex-set V, we write $\mathcal{H} \circ x = \sum_{S \in \mathcal{F}} \delta_x(S)$; we let $\mu(\mathcal{H})$ stand for the minimum of $\mathcal{H} \circ x$ taken over all incidence vectors x of tours through V.

All the cuts we ever find (and subsequently use) have the form

$$\mathcal{H} \circ x \geq \mu(\mathcal{H}). \tag{5.1}$$

Trivially, every subtour inequality has the form (5.1) with $\mathcal{H}$ having a single edge and $\mu(\mathcal{H}) = 0$. To see that comb inequalities have the form (5.1), let us call a hypergraph $\mathcal{H}$ a *comb* if $\mathcal{H}$ has edge-set $\{H, T_0, T_1, \ldots T_{2k}\}$ such that $k \geq 1$ and

- $T_0, T_1, \ldots T_{2k}$ are pairwise disjoint,
- H meets each of $T_0, T_1, \ldots T_{2k}$ but does not contain any.

Since (2.2) implies $x(S) = |S| - 1 - \delta(S)/2$, each comb inequality (3.1) of Section 3,

$$x(H) + \sum_{i=1}^{2k+1} x(T_i) \;\leq\; (|H|-1) + \sum_{i=1}^{2k+1} (|T_i| - 1) - k,$$

may be written as

$$\mathcal{H} \circ x \geq \; 2k. \tag{5.2}$$

Since (5.2) is satisfied by all incidence vectors x of tours, we have $\mu(\mathcal{H}) \geq 2k$; since there is a tour with incidence vector x such that $\mathcal{H} \circ x = 2k$, we have $\mu(\mathcal{H}) \leq 2k$. Hence $\mu(\mathcal{H}) = 2k$

and (5.2) is a special case of (5.1).

We store a number of the cuts we have used and refer to their collection as our *cutpool.* The cutpool, initially empty, may acquire new items with each new LP relaxation of our TSP instance. Each of these new relaxations arises from the relaxation preceding it by adding new cuts. (Actually, some of the constraints of the preceding relaxation may be also removed. The removal, although crucial for maintenance of sleek LP relaxations, is irrelevant to the cutpool, and so we shall not discuss it now.) Some, but not necessarily all, of these new cuts are added to the cutpool as soon as the new relaxation is solved. Specifically, each of the new cuts is added to the cutpool if and only if it satisfies all three of the following entrance requirements:

(i) it is *active* in the sense that, in the optimal solution of the dual of the new relaxation, the corresponding variable has a nonzero value,

(ii) it is not a subtour inequality,

(iii) it is not a *blossom inequality*
(meaning a comb inequality where each tooth consists of precisely two points).

We impose these entrance requirements only to save space: inequalities that fail (i) are useless at least for the moment, and inequalities that fail (ii) or (iii) can be recovered relatively quickly, should they become violated again in the future.

The notion of a cutpool was introduced by Padberg and Rinaldi [24, 26]; our cutpool is managed in a little different way (which we have just specified); it is used in a much different way (which is the subject of the next three sections), and it is stored in a much different way (which we are about to outline).

The cutpool consists of a number of inequalities that have the form (5.1). No matter how the cutpool is stored, its right-hand sides alone take up a negligible proportion of the total space; besides, every inequality $\mathcal{H}_i \circ x \geq \mu(\mathcal{H}_i)$ in our cutpool is such that, given $\mathcal{H}_i$, we can easily compute $\mu(\mathcal{H}_i)$. Hence storing the cutpool amounts to storing a family $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M\}$ of hypergraphs.

We take advantage of the empirical observation that distinct hypergraphs in our cutpool share edges relatively often: rather than storing each member of the cutpool independently from the other members, we first store the union $\mathcal{A}$ of all the edge-sets of the hypergraphs in the cutpool and then represent the edge-set of each $\mathcal{H}_i$ simply by pointers to the appropriate members of $\mathcal{A}$. Let us refer to $\mathcal{A}$ as our *archive*.

Our scheme for storing the archive has evolved from a chain of heuristic arguments. First, when a hypergraph $\mathcal{H}$ is admitted to the cutpool, our optimal solution x of the current LP relaxation satisfies $\mathcal{H} \circ x = \mu(\mathcal{H})$. Hence edges S of $\mathcal{H}$ seem likely to be tight or nearly tight with respect to x in the sense of having small $\delta_x(S)$: for instance, if $\mathcal{H}$ is a comb with $2k+1$ teeth then the value of $\delta_x(S)$ averaged over all the edges S of $\mathcal{H}$ comes to $k/(k+1)$. Second, even though x keeps changing as we keep adding new constraints to the LP relaxation, it seems likely that at least some of the sets S in our archive will persist in maintaining a reasonably small $\delta_x(S)$. Third, as the LP relaxation gets tighter and tighter, its optimal solution x seems to be more and more likely to closely approximate an optimal tour. Putting all these arguments together, we were led to suspect that many members of our archive may have fairly small values of $\delta_x(S)$ with respect to the incidence vector x of an optimal tour, and also with respect to the incidence vector x of a near-optimal tour delivered by a decent heuristic.

Now consider an arbitrary tour T, its incidence vector x, and an arbitrary subset S of V. When S consists of precisely k circular arcs of T, we have $\delta_x(S) = 2k$ with the same value of k; once all the vertices of T have been enumerated in their cyclic order, S can be represented simply by $2k$ pointers, one pointer to the beginning of each of the k circular arcs and one pointer to the end; if $2k$ is significantly smaller than $|S|$ then this representation is significantly more compact than the straightforward representation of S by a list of its elements.

That, with T being the near-optimal tour at our disposal, is how we represent every member of the archive. The heuristic reasoning that motivated this storage scheme in the beginning was confirmed by empirical evidence in the end. Our cutpool for TSPLIB problem pla7397 eventually grew to 2,888,447 hypergraphs with an archive of 33,814,752 sets; the average size of a set in the archive was around 200, but the average value of $\delta_x(S)$ was less than 11.

6 FINDING CUTS: CONSECUTIVE ONES

The purpose of this section is to describe one way in which the cutpool may help us in finding new cuts. Actually, here it is just the archive rather than the entire cutpool that is put to a good use. Throughout this section, we let `exterior` denote an arbitrary but fixed element of V and we write $W = V - \{\texttt{exterior}\}$ just as we did in Section 2. The central notion of the present section is this: a family $\mathcal{F}$ of subsets of a set W is said to have *the consecutive ones property* if W can be endowed with a linear order $\prec$ so that each S in $\mathcal{F}$ has the form $\{y \in W : a \preceq y \preceq b\}$ with $a, b \in W$. On the one hand, it is easy to see that a family $\mathcal{F}$ of subsets of our W has the consecutive ones property if and only if $\mu(\mathcal{H}) = 0$, where $\mathcal{H} = (V, \mathcal{F})$; on the other hand, Booth and Lueker [2] proved that a family $\mathcal{F}$ of sets has the consecutive ones property if and only if there is a PQ-tree T' with $\mathcal{B}(T') \supseteq \mathcal{F}$. In this way, the consecutive ones property provides a link between the TSP and PQ-trees.

Here is our starting point:

THEOREM 6.1 *There is a polynomial-time algorithm that, given any x with properties (2.1), (2.2), given any PQ-tree T compatible with x and given any subset S of W such that*
(a) $\delta(S) < 2$,
(b) no PQ-tree T' satisfies $\mathcal{B}(T') \supseteq \mathcal{B}(T) \cup \{S\}$,
returns either a comb inequality (with precisely three teeth) violated by x or a subtour inequality violated by x. □

One application of this theorem is obvious: with x being our optimal solution of the current LP relaxation and with T being the finest PQ-tree at our disposal that is compatible with x, we simply scan our archive $\mathcal{A}$, looking for its members S that satisfy both (a) and (b). Each such S yields a new cut. What our program actually does is more complicated and more productive; for clarity, we shall describe it in a more general setting.

An *independence system* is an ordered pair $(X, \mathcal{I})$ such that X is a finite set and $\mathcal{I}$ is a nonempty collection of subsets of X that is closed under taking subsets (that is to say, if $R \subset S$ and $S \in \mathcal{I}$ then $R \in \mathcal{I}$). Members of $\mathcal{I}$ are referred to as *independent* and the remaining subsets of X are referred to as *dependent*; minimal (with respect to set-inclusion) dependent sets are called *circuits*. An independence system may be presented in a variety of ways; one of them is an *oracle* that, given any subset S of X, answers correctly the query "Is S independent?".

In the independence systems that we are concerned with here, elements of X are sets and a subset of X is independent if, and only if, it has the consecutive ones property. An oracle which presents these independence systems is the Booth-Lueker algorithm described in the Section 4: as mentioned at the beginning of the present section, a family $\mathcal{F}$ of sets has the consecutive ones property if and only if there is a PQ-tree T' with $\mathcal{B}(T') \supseteq \mathcal{F}$. Trivially, all two-point subsets of X are independent in these independence systems; it is easy to see that a three-point subset $\{S_1, S_2, S_3\}$ of X is a circuit if and only if

$$\text{all three of } (S_1 \cap S_2) - S_3, (S_1 \cap S_3) - S_2, (S_2 \cap S_3) - S_1 \text{ are nonempty,} \tag{6.1}$$

or else

$$\text{all four of } S_1 \cap S_2 \cap S_3, S_1 - (S_2 \cup S_3), S_2 - (S_1 \cup S_3), S_3 - (S_1 \cup S_2) \text{ are nonempty.} \tag{6.2}$$

Now we are going to characterize all the circuits of size at least four in these independence systems; for this purpose, we need a few definitions.

We shall say that $\{S_1, S_2, S_3, S_4\}$ is a *family of type 0* if there are pairwise disjoint nonempty sets A_1, A_2, A_3, A_4, B such that, with subscript arithmetic modulo 4, each S_i is $A_i \cup A_{i+1} \cup B$.

We shall say that $\{S_1, S_2, \ldots, S_m\}$ is a *family of type I* if $m \geq 4$ and there are pairwise disjoint sets $A_1, C_1, A_2, C_2, \ldots, A_m, C_m$ such that $A_1, A_2, \ldots, A_m$ are nonempty and, with subscript arithmetic modulo m, each S_i is $A_i \cup C_i \cup A_{i+1}$.

We shall say that $\{S_1, S_2, \ldots, S_m, T_1, T_2\}$ is a *family of type II* if $m \geq 2$ and there are pairwise disjoint sets $A_0, C_1, A_1, C_2, A_2, \ldots, A_{m-1}, C_m, A_m, B$ such that $A_0, A_1, A_2, \ldots, A_m, B$ are nonempty, each S_i is $A_{i-1} \cup C_i \cup A_i$, and $C_1, A_1, C_2, A_2, \ldots, A_{m-1}, C_m, B \subset T_1 \cap T_2$, $A_0 \subseteq T_1 - T_2$, $A_m \subseteq T_2 - T_1$.

We shall say that $\{S_1, S_2, \ldots, S_m, T\}$ is a *family of type III* if $m \geq 3$ and there are pairwise disjoint sets $A_0, C_1, A_1, C_2, A_2, \ldots, A_{m-1}, C_m, A_m, B$ such that $A_0, A_1, A_2, \ldots, A_m, B$ are nonempty, each S_i is $A_{i-1} \cup C_i \cup A_i$, and $C_1, A_1, C_2, A_2, \ldots, A_{m-1}, C_m, B \subset T$, $A_0 \cap T = \emptyset$, $A_m \cap T = \emptyset$.

We shall say that $\{H, T_1, T_2, T_3\}$ is a *family of type IV* if T_1, T_2, T_3 are pairwise disjoint and if H meets each of T_1, T_2, T_3 but does not contain any.

We shall say that $\{H, T_1, T_2, T_3\}$ is a *family of type V* if $T_1 \cap T_2 = \emptyset$, $T_1 \cup T_2 \subseteq T_3$, $H \cap T_1 \neq \emptyset$, $H \cap T_2 \neq \emptyset$, $H \not\supseteq T_1$, $H \not\supseteq T_2$, $H \not\subseteq T_3$.

THEOREM 6.2 *Let $\mathcal{F}$ be a family of at least four sets such that $\mathcal{F}$ lacks the consecutive ones property but every proper subfamily of $\mathcal{F}$ has the consecutive ones property. Then $\mathcal{F}$ is a family of type 0, I, II, III, IV, or V.* □

Next, let us consider the following problem:

($*$) Given an oracle specifying an independence system $(X, \mathcal{I})$,
given a nonnegative weight assigned to each point of X, and
given a positive threshold t,
find circuits whose weights
(with the weight of a set defined as the sum of the weights of its elements)
are strictly below the threshold t.

We are interested in the special case of ($*$) where $(X, \mathcal{I})$ is an independence system of our kind, the weight of each S' in X is $\delta(S')$, and $t = 2$; here is why. By Theorem 6.2, each solution of this special case of ($*$) is a family $\mathcal{F}$ of type 0, I, II, III, IV, or V such that $\sum_{S \in \mathcal{F}} \delta(S) < 2$. If $\mathcal{F}$ turns out to be of type IV then x violates the comb inequality $\mathcal{H} \circ x \geq 2$ with $\mathcal{H}$ having handle H and teeth T_1, T_2, T_3; if $\mathcal{F}$ turns out to be of type V then x violates the comb inequality $\mathcal{H} \circ x \geq 2$ with $\mathcal{H}$ having handle H and teeth $T_1, T_2, V - T_3$; the following theorem takes care of the remaining cases.

THEOREM 6.3 *There is a polynomial-time algorithm that, given any family $\mathcal{F}$ of subsets of W such that*

- *$\mathcal{F}$ lacks the consecutive ones property,*
- *every proper subfamily of $\mathcal{F}$ has the consecutive ones property,*
- *$\mathcal{F}$ is neither of type IV nor of type V,*

and given any vector x with properties (2.1), (2.2), and

- *$\sum_{S \in \mathcal{F}} \delta(S) < 2$,*

returns a subtour inequality violated by x. □

Our heuristic for attacking problem ($*$) uses a function `fetch` that, given any independent set Q sorted in a nondecreasing order of weights and given a point s such that $Q \cup \{s\}$ is dependent, attempts to find a circuit C of weight less than t such that $C \subseteq Q \cup \{s\}$. This function treats Q as a queue; its prototype goes as follows:

```
$C = \{s\}$;
add $s$ to the front of $Q$;
while $C$ is independent and its weight is less than $t$
do    replace $Q$ with its shortest dependent prefix;
      add the last element of $Q$ to $C$ and move it to the front of $Q$;
end
if the weight of $C$ is less than $t$ then return $C$ else return a failure message;
```

To see that C is a circuit, note that a point p gets included in C only if $Q - \{p\}$ has been found independent for some superset Q of C.

Actually, we use a modification of this prototype. The reason is that we are not collecting lightweight circuits: what we are collecting are cuts obtained from these circuits either directly (in case the circuit is a family of type IV or V) or by the algorithm of Theorem 6.3. The cut can be a comb inequality only if `fetch` returns a circuit within three iterations of its while loop; in all other cases, we are in for a potentially long computation with the uncertain prospect of finding a subtour inequality violated by x. Since subtour inequalities violated by x are relatively easy to spot by direct methods, we curtail the while loop of `fetch` by exiting as soon as C has acquired its fourth element. Here is the resulting version of `fetch` with our PQ-tree oracle explicitly built in (and s switched to S):

```
$C = \{S\}$, $w = \delta(S)$;
while $|C| < 4$, $w < 2$, and there is a PQ-tree $T'$ with $\mathcal{B}(T') \supseteq C$
do    $T'$ = the coarsest PQ-tree with $\mathcal{B}(T') \supseteq C$
      $S$ = the first set in the sequence $Q$;
      while there is a PQ-tree $T''$ with $\mathcal{B}(T'') \supseteq \mathcal{B}(T') \cup \{S\}$
      do    replace $T'$ with the coarsest such $T''$;
            replace $S$ with the set following it in the sequence $Q$;
      end
      move $S$ from $Q$ to $C$ and increment $w$ by $\delta(S)$;
end
if $C$ yields a cut then return this cut else return a failure message;
```

A prototype of the heuristic for attacking (*) goes as follows:

```
initialize an empty list $\mathcal{L}$ of lightweight circuits;
sort $X$ in a nondecreasing order of weights;
while $X$ is dependent
do    $Q$ = longest independent prefix of $X$;
      $s$ = the element of $X$ that follows $Q$;
      $C$ = fetch($Q, s$);
      if $C \neq$ failure message then add $C$ to $\mathcal{L}$;
      remove $s$ from $X$;
end
```

Again, we use a modification of this prototype. The reason is that, as s is getting closer and closer to the end of X, it is becoming more and more likely that `fetch` will return a failure message after only one iteration of its while loop, when C turns out to be too heavy even though its size is only two. The following variant avoids these useless calls of `fetch`:

```
initialize an empty list $\mathcal{L}$ of lightweight circuits;
sort $X$ in a nondecreasing order of weights as $s_1, s_2, \ldots, s_M$;
$Q =$ the empty sequence;
$i = 0$, $j = M + 1$;
while $i < j - 1$
do    if weight$(s_{i+1})$ + weight$(s_{j-1}) < t$
      then increment $i$;
           if $Q \cup \{s_i\}$ is independent
           then add $s_i$ to the end of $Q$;
           else $C =$ fetch$(Q, s_i)$;
                if $C \neq$ failure message then add $C$ to $\mathcal{L}$;
           end
      else decrement $j$;
           if $Q \cup \{s_j\}$ is dependent
           then $C =$ fetch$(Q, s_j)$;
                if $C \neq$ failure message then add $C$ to $\mathcal{L}$;
           end
      end
end
```

As in the application of Theorem 6.1, let T denote any PQ-tree compatible with x and let $\mathcal{A}$ denote the archive; we choose a subset of $X = \mathcal{B}(T) \cup \mathcal{A}$ for our X. Setting $X = \mathcal{B}(T) \cup \mathcal{A}$ could be a bad choice: an excessively large X can slow down all calls of `fetch` and, since each Q-node with d children contributes $\binom{d}{2}$ sets to $\mathcal{B}(T)$, the size of $\mathcal{B}(T)$ can easily reach a quadratic function of n. We choose $X = \mathcal{B}' \cup \mathcal{A}$ for some reasonably small subfamily $\mathcal{B}'$ of $\mathcal{B}(T)$ such that

$$\text{every PQ-tree } T' \text{ with } \mathcal{B}(T') \supseteq \mathcal{B}' \text{ satisfies } \mathcal{B}(T') \supseteq \mathcal{B}(T) \tag{6.3}$$

Depending on how we came by T, we may even have such a $\mathcal{B}'$ readily available; a good default choice of $\mathcal{B}'$ is the family $\mathcal{B}_0(T)$ that consists of

- all sets $D(u)$ such that u is a node of T, and
- all sets $D(u_i) \cup D(u_{i+1})$ such that $u_1, u_2, \ldots, u_d$ in this order are the children of some Q-node
  of T and $1 \leq i < d$.

To see $\mathcal{B}_0(T)$ satisfies (6.3) in place of $\mathcal{B}'$, observe that $R \cup S \in \mathcal{B}(T')$ whenever $R, S \in \mathcal{B}(T')$ and $R \cap S \neq \emptyset$.

Some of the expensive calls of `fetch` can be avoided altogether by relying on the algorithm of Theorem 6.1 (which is faster than `fetch`) whenever S satisfies assumptions (a), (b) of that theorem. The resulting improved version of our algorithm goes as follows.

```
initialize an empty list $\mathcal{L}$ of cuts;
$\mathcal{A}' = \emptyset$;
for all members $S$ of $\mathcal{A}$
do if $\delta(S) < 2$
    then if there is no PQ-tree $T'$ with $\mathcal{B}(T') \supseteq \mathcal{B}(T) \cup \{S\}$
          then add the resulting cut to $\mathcal{L}$;
          else add $S$ to $\mathcal{A}'$;
          end
    end
end

sort $\mathcal{A}'$ in a nondecreasing order of weights as $S_1, S_2, \ldots, S_M$;
choose a reasonably small $\mathcal{B}'$ that satisfies (6.3);
$Q = \mathcal{B}'$ in an arbitrary order;
$i = 0$, $j = M + 1$;
while $i < j - 1$
do    if $\delta(S_{i+1}) + \delta(S_{j-1}) < 2$
      then increment $i$;
           if there is a PQ-tree $T'$ with $\mathcal{B}(T') \supseteq \mathcal{B}(T) \cup \{S_i\}$
           then replace $T$ with the coarsest such $T'$;
                add $S_i$ to the end of $Q$;
           else $C = \texttt{fetch}(Q, S_i)$;
               if $C \neq$ failure message then add $C$ to $\mathcal{L}$;
           end
      else decrement $j$;
           if there is no PQ-tree $T'$ with $\mathcal{B}(T') \supseteq \mathcal{B}(T) \cup \{S_j\}$
           then $C = \texttt{fetch}(Q, S_j)$;
                if $C \neq$ failure message then add $C$ to $\mathcal{L}$;
           end
      end
end
```

The remainder of this section is devoted to proofs of the three theorems (in a permuted order).

PROOF OF THEOREM 6.2. Tucker ([29], Theorem 9) characterized families without the consecutive ones property as families $\mathcal{F}$ with at least one of the following five properties:

(I) There are a set R consisting of distinct elements $a_1, a_2, \ldots, a_m$ and sets $S_1, S_2, \ldots, S_m$ in $\mathcal{F}$ such that $m \geq 3$ and (with subscript arithmetic modulo m) $S_i \cap R = \{a_i, a_{i+1}\}$ for all i.

(II) There are a set R consisting of distinct elements $a_0, a_1, a_2, \ldots, a_m, b$ and sets $S_1, S_2, \ldots, S_m$, T_1, T_2 in $\mathcal{F}$ such that $m \geq 2$, $S_i \cap R = \{a_{i-1}, a_i\}$ for all i, and $T_1 \cap R = \{a_0, a_1, \ldots, a_{m-1}, b\}$, $T_2 \cap R = \{a_1, a_2, \ldots, a_m, b\}$.

(III) There are a set R consisting of distinct elements $a_0, a_1, a_2, \ldots, a_m, b$ and sets $S_1, S_2, \ldots, S_m$, T in $\mathcal{F}$ such that $m \geq 2$, $S_i \cap R = \{a_{i-1}, a_i\}$ for all i, and $T \cap R = \{a_1, a_2, \ldots, a_{m-1}, b\}$.

(IV) There are a set R consisting of distinct elements $a_1, a_2, a_3, b_1, b_2, b_3$ and sets H, T_1, T_2, T_3 in $\mathcal{F}$ such that $H \cap R = \{a_1, a_2, a_3\}$ and $T_i \cap R = \{a_i, b_i\}$ for all i.

(V) There are a set R consisting of distinct elements a_1, a_2, a_3, b_1, b_2 and sets H, T_1, T_2, T_3 in $\mathcal{F}$ such that $H \cap R = \{a_1, a_2, a_3\}$ and $T_1 \cap R = \{a_1, b_1\}$, $T_2 \cap R = \{a_2, b_2\}$, $T_3 \cap R = \{a_1, a_2, , b_1, b_2\}$.

Consider an arbitrary family $\mathcal{F}$ of at least four sets such that $\mathcal{F}$ satisfies one of (I), (II), (III), (IV), (V), but no proper subfamily of $\mathcal{F}$ satisfies any of (I), (II), (III), (IV), (V). We shall prove that

$\mathcal{F}$ is of type 0 or X if it satisfies (X) with X=I or II

and that

$\mathcal{F}$ is of type X if it satisfies (X) with X=III or IV or V.

For this purpose, let $\mathcal{F}(u)$ denote the family of all the members of $\mathcal{F}$ that include point u; in each of Tucker's five cases, let c denote an arbitrary point (if any exists) such that $\mathcal{F}(c)$ is distinct from all $\mathcal{F}(v)$ with $v \in R$.

Case I: Here, $|\mathcal{F}(c)| \leq 1$ or $\mathcal{F}(c) = \mathcal{F}$. [Else symmetry allows us to assume that $S_1 \in \mathcal{F}(c)$, $S_2 \notin \mathcal{F}(c)$. Now consider the smallest j other than 1 such that $S_j \in \mathcal{F}(c)$. Since $\mathcal{F}(c) \neq \mathcal{F}(a_m)$, we have $j < m$; condition (I) is met by j in place of m and by c in place of a_1.]

Subcase I.0: There is a c with $\mathcal{F}(c) = \mathcal{F}$.

Here, $\mathcal{F}$ is of type 0: we have $m = 4$ [else condition (III) with $m = 2$ is met by S_4 in place of T and by a_1, c, a_3, a_4 in place of a_0, a_1, a_2, b] and there is no u with $|\mathcal{F}(u)| = 1$ [else symmetry allows us to assume $\mathcal{F}(u) = \{S_1\}$, in which case condition (III) with $m = 2$ is met by S_4 in place of T and by u, c, a_3, a_4 in place of a_0, a_1, a_2, b].

Subcase I.1: There is no c with $\mathcal{F}(c) = \mathcal{F}$.

Here, $\mathcal{F}$ is of type I.

Case II: Here, there are subscripts i and k such that $S_j \in \mathcal{F}(c)$ if and only if $i \leq j \leq k$. [Else there are subscripts i and k such that $i < k-1$, $S_i, S_k \in \mathcal{F}(c)$ and $S_j \notin \mathcal{F}(c)$ whenever $i < j < k$. But then condition (I) is met by $S_i, S_{i+1}, \ldots, S_k$ with $R = \{c, a_i, a_{i+1}, \ldots, a_{k-1}\}$.] Symmetry allows us to distinguish between three subcases.

Subcase II.0: $T_1 \notin \mathcal{F}(c)$, $T_2 \notin \mathcal{F}(c)$.

Here, $\mathcal{F}(c) = \emptyset$ or else $m = 2$ and $\mathcal{F}(c) = \{S_1, S_2\}$. [Assume the contrary. Now if there is a subscript j with $i \leq j \leq k$ and $1 < j < m$ then condition (III) with $m = 2$ is met by S_j, T_1, T_2 in place of S_1, S_2, T and by c, a_j, a_0, a_m in place of a_0, a_1, a_2, b; else symmetry allows us to assume that $i = k = m$, in which case condition (III) is met by substituting c for a_m and T_2 for T.]

If $m = 2$ and $\mathcal{F}(c) = \{S_1, S_2\}$ then $\mathcal{F}$ is of type 0 since we find ourselves in Subcase I.0 with a_0, c, a_2, b in place of a_1, a_2, a_3, a_4, with S_1, S_2, T_2, T_1 in place of S_1, S_2, S_3, S_4, and with a_1 in place of c.

Subcase II.1: $T_1 \in \mathcal{F}(c)$, $T_2 \notin \mathcal{F}(c)$.

Here, $\mathcal{F}(c) = \{T_1\}$. [Assume the contrary. If $k = m$ then condition (I) with $m = 3$ is met by S_m, T_1, T_2 in place of S_1, S_2, S_3 and by a_m, c, b in place of a_1, a_2, a_3; if $1 < k < m$ then condition (II) is met by $S_k, S_{k+1}, \ldots, S_m, T_1, T_2$ with $R = \{c, a_k, a_{k+1}, \ldots, a_m, b\}$; if $k = 1$ then $\mathcal{F}(c) = \mathcal{F}(a_0)$.]

Subcase II.2: $T_1 \in \mathcal{F}(c)$, $T_2 \in \mathcal{F}(c)$.

Here, $\mathcal{F}(c) = \{S_i, T_1, T_2\}$. [Since $\mathcal{F}(b) = \{T_1, T_2\}$, we must have $i \leq k$. If $i = k-1$ then $\mathcal{F}(c) = \mathcal{F}(a_i)$; if $i < k-1$ then condition (II) is met when all S_j with $i < j < k$ are removed and c is substituted for the sequence $a_i, a_{i+1}, \ldots, a_{k-1}$.]

Case III: As in Case II, there are subscripts i and k such that $S_j \in \mathcal{F}(c)$ if and only if $i \leq j \leq k$.

Subcase III.0: $T \notin \mathcal{F}(c)$.

Here, $\mathcal{F}(c) = \emptyset$. [Assume the contrary. If $k = 1$ then $\mathcal{F}(c) = \mathcal{F}(a_0)$. If $1 < k < m$ then condition (III) is met by $S_k, S_{k+1}, \ldots, S_m, T$ with $R = \{c, a_k, a_{k+1}, \ldots, a_m, b\}$. If $i = k = m$ then $\mathcal{F}(c) = \mathcal{F}(a_m)$. If $i = m-1$ and $k = m$ then condition (III) is met when S_m is removed and a_{m-1}, a_m are replaced with c; if $i < m-1$ and $k = m$ then condition (I) with $m = 3$ is met by S_{m-2}, S_m, T in place of S_1, S_2, S_3 and a_{m-2}, c, a_{m-1} in place of a_1, a_2, a_3.]

Subcase III.1: $T \in \mathcal{F}(c)$.

Here, $\mathcal{F}(c) = \{S_i, T\}$. [Since $\mathcal{F}(b) = \{T\}$, we must have $i \leq k$. If $i = k-1$ then $\mathcal{F}(c) = \mathcal{F}(a_i)$; if $i < k-1$ then condition (III) is met when all S_j with $i < j < k$ are removed and c is substituted for the sequence $a_i, a_{i+1}, \ldots, a_{k-1}$.]

Case IV: Here, $\mathcal{F}$ is of type IV: we have $\mathcal{F}(c) \subseteq \{H\}$ [else condition (I) with $m = 3$ or condition (III) with $m = 2$ is met].

Case V: Here, $\mathcal{F}$ is of type V: we have $T_1 \notin \mathcal{F}(c)$, $T_2 \notin \mathcal{F}(c)$ [else condition (I) with $m = 3$ or condition (III) with $m = 2$ is met]. □

A *set-function* is a real-valued function defined on all subsets of some set; a set-function f defined on all subsets of V is called *submodular* if $f(R\cap S)+f(R\cup S)\le f(R)+f(S)$ for all choices of subsets R,S of V. To see that

$$\delta \text{ is a submodular set-function,} \qquad (6.4)$$

observe (as in the proof of Lemma 2.1) that $\delta(R\cap S)+\delta(R\cup S)=\delta(R)+\delta(S)-2x(R-S,S-R)$. It is a routine matter to verify that

$$\delta(S_1\cup S_2\cup S_3)+\delta((S_1\cap S_2)-S_3)+\delta((S_1\cap S_3)-S_2)+\delta((S_2\cap S_3)-S_1)\le\delta(S_1)+\delta(S_2)+\delta(S_3)-2; \qquad (6.5)$$

substituting $V-S_i$ for each S_i in (6.5), we obtain

$$\delta(S_1\cap S_2\cap S_3)+\delta(S_1-(S_2\cup S_3))+\delta(S_2-(S_1\cup S_3))+\delta(S_3-(S_1\cup S_2))\le\delta(S_1)+\delta(S_2)+\delta(S_3)-2. \qquad (6.6)$$

Properties (6.4), (6.5), (6.6) are the only properties of δ that we rely on in our next proof.

PROOF OF THEOREM 6.3. First, let us dispose of the case where $\mathcal{F}$ includes precisely three sets. If (6.1) holds then (6.5) guarantees that at least one of $S_1\cup S_2\cup S_3$, $(S_1\cap S_2)-S_3$, $(S_1\cap S_3)-S_2$, $(S_2\cap S_3)-S_1$ is bad; if (6.2) holds then (6.6) guarantees that at least one of $S_1\cap S_2\cap S_3$, $S_1-(S_2\cup S_3)$, $S_2-(S_1\cup S_3)$, $S_3-(S_1\cup S_2)$ is bad.

Now we may assume that $\mathcal{F}$ includes more than three sets; here, Theorem 6.2 guarantees that $\mathcal{F}$ is a family of type 0, I, II, or III. Finding out the type of $\mathcal{F}$ is a trivial matter; to spell out its details, let the *degree* of a point mean the number of members of $\mathcal{F}$ in which this point is included, and let $D(\mathcal{F})$ denote the set of degrees of all the points in $\cup_{S\in\mathcal{F}}S$. Since

$$\begin{array}{rcll}
 & D(\mathcal{F}) & =\{2,4\} & \text{whenever } \mathcal{F} \text{ is of type 0,}\\
\{2\}\subseteq & D(\mathcal{F}) & \subseteq\{1,2\} & \text{whenever } \mathcal{F} \text{ is of type I,}\\
\{2,4\}\subseteq & D(\mathcal{F}) & \subseteq\{1,2,3,4\} & \text{whenever } \mathcal{F} \text{ is of type II,}\\
\{1,3\}\subseteq & D(\mathcal{F}) & \subseteq\{1,2,3\} & \text{whenever } \mathcal{F} \text{ is of type III,}
\end{array}$$

the type of $\mathcal{F}$ is determined by $D(\mathcal{F})$ except when $D(\mathcal{F})=\{2,4\}$; in the exceptional case, $\mathcal{F}$ is of type 0 if and only if $|\mathcal{F}|=4$ and no member of $\mathcal{F}$ contains another. Once the type of $\mathcal{F}$ has been determined, labeling the members of $\mathcal{F}$ properly (as S_1,S_2,S_3,S_4 if $\mathcal{F}$ is of type 0, as $S_1,S_2,\ldots,S_m$ if $\mathcal{F}$ is of type I, as $S_1,S_2,\ldots,S_m,T_1,T_2$ if $\mathcal{F}$ is of type II, and as $S_1,S_2,\ldots,S_m,T$ if $\mathcal{F}$ is of type III) is another trivial matter.

If $\mathcal{F}$ is of type 0, I, or III then the following algorithm either returns a bad set or reduces $\mathcal{F}$ to a family with three sets that lacks the consecutive ones property.

```
while $\mathcal{F}$ includes more than three sets
do  if $S_{m-1}\cap S_m$ is bad
    then return this bad set;
    else replace $S_m$ with $S_{m-1}\cup S_m$ and then decrement $m$;
end
```

Submodularity of δ guarantees that the invariant $\sum_{S\in\mathcal{F}} \delta(S) < 2$ is maintained throughout.

If $\mathcal{F}$ is of type II then we set $T = T_1 \cap T_2$ and replace T_1, T_2 with T; unless $T_1 \cup T_2$ is bad, submodularity of δ guarantees that $\sum_{i=1}^{m} \delta(S_i) + \delta(T) < 2$. If $m \geq 3$ then $\{S_1, S_2, \ldots, S_m, T\}$ is of type III; else it has precisely three sets and lacks the consecutive ones property. $\square$

PROOF OF THEOREM 6.1. If (b) holds then the polynomial-time algorithm at the end of Section 4 returns one of the following:

(i) a node u of T and children v_1, v_2, v_3 of u such that
S meets each of $D(v_1), D(v_2), D(v_3)$ but does not contain any.

(ii) a node u of T and children v_1, v_2 of u such that
$S \cap D(v_1) \neq \emptyset,\ \ S \cap D(v_2) \neq \emptyset,\ \ S \not\supseteq D(v_1),\ \ S \not\supseteq D(v_2),\ \ S \not\subseteq D(u).$

(iii) a Q-node of T with children $u_1, u_2, \ldots, u_d$ in the left-to-right order
and subscripts i, j, k with $i < j < k$ such that
$S \cap D(u_i) \neq \emptyset,\ \ S \not\supseteq D(u_j),\ \ S \cap D(u_k) \neq \emptyset.$

(iv) a Q-node u of T with children $u_1, u_2, \ldots, u_d$ in the left-to-right order such that
$S \not\subseteq D(u),\ \ S \not\supseteq D(u_1),\ \ S \not\supseteq D(u_d),\ \ S \cap (\cup_{1<t<d} D(u_t)) \neq \emptyset.$

If the algorithm returns (i) then the comb $\mathcal{H}$ with handle S and teeth $D(v_1), D(v_2), D(v_3)$ satisfies $\mathcal{H} \circ x = \delta(S)$; if the algorithm returns (ii) then the comb $\mathcal{H}$ with handle S and teeth $V - D(u)$, $D(v_1), D(v_2)$ satisfies $\mathcal{H} \circ x = \delta(S)$; in either of these two cases, assumption (a) guarantees that the comb inequality is violated by x.

If (iii) holds then set $\mathcal{F} = \{\cup_{t\leq j} D(u_t), \cup_{t\geq j} D(u_t), S\}$; if (iv) holds then set $\mathcal{F} = \{\cup_{t<d} D(u_t), \cup_{t>1} D(u_t), S\}$; in either of these two cases, assumption (a) guarantees that $\mathcal{F}$ and x satisfy the assumptions of Theorem 6.3. $\square$

7 FINDING CUTS: GLUING

In the last section, we described one way in which the cutpool may help us in finding new cuts; the purpose of the present section is to describe another way, one which amounts to "gluing together" old constraints into a conglomerate cut. Grötschel and Pulleyblank [16] used a notion of gluing in the analysis of their *clique tree inequalities*. (These inequalities are satisfied by incidence vectors of all tours; their class includes all the subtour inequalities and all the comb inequalities; in fact, the class of all subtour inequalities and all comb inequalities is just a speck in the universe of clique tree inequalities.) On the one hand, we glue in a way that is slightly more restricted than the Grötschel-Pulleyblank way; on the other hand, we start out with a wider class of elementary building blocks. The resulting recursively defined class of *clique-tree-like inequalities* nearly but not quite subsumes the class of clique tree inequalities; to reduce confusion, we postpone all discussion of clique tree inequalities and their relationship to clique-tree-like inequalities till the very end of this section.

Unlike Grötschel and Pulleyblank, we use gluing not only as an *analytical* tool but also as an *algorithmic* tool; it is the algorithmic use of gluing that is the focal point of this section.

To define clique-tree-like inequalities, we shall first define *clique-tree-like hypergraphs*, and we shall do that recursively. Two special kinds of hypergraphs serve as elementary blocks in building up clique-tree-like hypergraphs. A *comb* is a hypergraph with vertex-set V and edge-set $\{H, T_0, T_1, \dots T_{2k}\}$ such that $k \geq 1$ and

- $T_0, T_1, \dots T_{2k}$ are pairwise disjoint,
- H meets each of $T_0, T_1, \dots T_{2k}$ but does not contain any;

a *flipped comb* is a hypergraph with vertex-set V and edge-set $\{H, T_0, T_1, \dots T_{2k}\}$ such that $k \geq 1$ and

- $H - T_0 \neq \emptyset$, $H \cup T_0 \neq V$, and $T_0 \supseteq T_i$ for all $i = 1, 2, \dots, 2k$,
- H meets each of $T_1, \dots T_{2k}$ but does not contain any,
- $T_1, \dots T_{2k}$ are pairwise disjoint;

in combs and flipped combs, H is called a *handle* and $T_0, T_1, \dots T_{2k}$ are called *teeth.* We say that a hypergraph $\mathcal{H}$ arises from hypergraphs $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_m$ by *gluing along edge* C if

- $\mathcal{H}_i = (V, \mathcal{F}_i)$ for all i and $\mathcal{H} = (V, \cup_{i=1}^m \mathcal{F}_i)$,
- C belongs to each $\mathcal{F}_i$,
- $A \cap B = \emptyset$ whenever $A \in \mathcal{F}_i - \{C\}$, $B \in \mathcal{F}_j - \{C\}$, and $i \neq j$,
- some vertex in C belongs to no edge of $\mathcal{H}$ other than C,
- some vertex in V belongs to no edge of $\mathcal{H}$.

A *clique-tree-like hypergraph* is any hypergraph $\mathcal{H}$, with edge-set partitioned into a set of handles and a set of teeth, such that either

- $\mathcal{H}$ is a comb or a flipped comb,

or else

- $\mathcal{H}$ arises from at least two clique-tree-like hypergraphs by gluing along a tooth.

THEOREM 7.1 *If $\mathcal{H}$ is a clique-tree-like hypergraph with t teeth then $\mu(\mathcal{H}) = t - 1$.* □

By a *clique-tree-like inequality*, we mean any inequality

$$\mathcal{H} \circ x \geq t - 1$$

such that $\mathcal{H}$ is a clique-tree-like hypergraph with t teeth; Theorem 7.1 guarantees that all clique-tree-like inequalities are satisfied by all incidence vectors x of tours.

Before proving Theorem 7.1, let us describe the way we use it. We sweep through the archive; given each new set C from the archive, we search for hypergraphs in our cutpool that can be glued together along their common tooth C into a conglomerate hypergraph $\mathcal{H}$ such that the optimal solution x of our most recent LP relaxation satisfies $\mathcal{H} \circ x < \mu(\mathcal{H})$. For this purpose, let $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$ denote all the hypergraphs in the cutpool that contain C as a tooth; let us write $\mathcal{H}_i = (V, \mathcal{F}_i)$ for all i. We want to find a subset S of $\{1, 2, \ldots, M\}$ such that

(i) $A \cap B = \emptyset$ whenever $A \in \mathcal{F}_i - \{C\}$, $B \in \mathcal{F}_j - \{C\}$, and $i \in S$, $j \in S$, $i \neq j$,

and, with $\mathcal{H} = (V, \cup_{i \in S} \mathcal{F}_i)$,

(ii) some vertex in C belongs to no edge of $\mathcal{H}$ other than C,

(iii) some vertex in V belongs to no edge of $\mathcal{H}$,

(iv) $\mathcal{H} \circ x < \mu(\mathcal{H})$.

The problem of finding such a set S may be cast in the more general setting of independence systems (see Section 5 for the standard definition): given an independence system $(X, \mathcal{I})$, given an assignment of nonnegative weights a_i to points i of X, and given a nonnegative number b, find an independent set S that satisfies

(v) $b + \sum_{i \in S}(a_i - b) < 0$

or show that no such S exists. To see that our particular problem fits this general paradigm, write $X = \{1, 2, \ldots, M\}$ and set $S \in \mathcal{I}$ if and only if S satisfies (i), (ii), (iii). Trivially, $(X, \mathcal{I})$ is an independence system: $\emptyset \in \mathcal{I}$ and $\mathcal{I}$ is closed under taking subsets. Now write

$$a_i = \mathcal{H}_i \circ x - \mu(\mathcal{H}_i), \qquad b = \delta_x(C),$$

and let t_i denote the number of teeth in $\mathcal{H}_i$. Theorem 7.1 guarantees that $\mu(\mathcal{H}_i) = t_i - 1$ for all i and that, as long as (i), (ii), (iii) are satisfied, $\mu(\mathcal{H}) = \sum_{i \in S}(t_i - 1)$. Since

$$\mathcal{H} \circ x = b + \sum_{i \in S}(a_i + \mu(\mathcal{H}_i) - b),$$

(iv) and (v) are equivalent whenever S is independent.

Let us describe our search algorithm in the general setting of independence systems. To begin, we may remove from X all points j such that $a_j > b$: removal of all such points transforms each independent set S into an independent set R with $\sum_{i \in R}(a_i - b) \leq \sum_{i \in S}(a_i - b)$. After this clean-up, let us write $X = \{1, 2, \ldots, M\}$ and, for future reference,

$$c_j = \sum_{i=j+1}^{M} (a_i - b).$$

The algorithm itself is recursive. Its idea may be easiest to grasp in terms of the *enumeration tree*, a complete binary tree of depth M, whose 2^j nodes at level j are in a one-to-one correspondence with the 2^j subsets of $\{1, 2, \ldots, j\}$: if a node at level $j-1$ corresponds to a subset R of $\{1, 2, \ldots, j-1\}$ then its left child corresponds to R and its right child corresponds to $R \cup \{j\}$. This tree gets *pruned* by removing all the descendants of a node whenever it is blatantly obvious that none of these descendants corresponds to an independent set S with property (v). Here, "blatantly obvious" can mean one of two different things. First, if a node corresponds to a set R that is not independent then no superset of R is independent, and so we may safely disregard all the descendants of this node; second, if a node at level j corresponds to a set R such that $b + \sum_{i \in R}(a_i - b) + c_j \geq 0$ then no superset S of R has property (v), and so we may safely disregard all the descendants of this node. Recursive calls of the algorithm are in a one-to-one correpondence with the nodes of the resulting pruned enumeration tree except for leaves at level M: each of these leaves that persists in the pruned tree corresponds to an independent set S with property (v).

Common sense seems to suggest that our chances of discovering at least one independent set S with property (v) increase as the value of b increases; for this reason, we sweep through the archive in a nonincreasing order of $\delta_x(C)$. Independently of this stratagem, the algorithm as described so far may come across a member C of the archive that yields an overwhelmingly large number of independent sets S with property (v); to stop this kind of good luck from turning into a disaster, we terminate the search whenever it has produced 50 new cuts.

Next, let us get started on proving Theorem 7.1. To establish the lower bound, $\mu(\mathcal{H}) \geq t - 1$, we prove a more general theorem:

THEOREM 7.2 *If $\mathcal{H}$ arises from $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ by gluing along any common edge then $\mu(\mathcal{H}) \geq \sum_{i=1}^{m} \mu(\mathcal{H}_i)$.*

PROOF. We may restrict ourselves to the case of $m = 2$. Let C denote the common edge of $\mathcal{H}_1$ and $\mathcal{H}_2$; let us write $\mathcal{H}_i = (V, \mathcal{F}_i)$ and $\mathcal{H}'_i = (V, \mathcal{F}_i - \{C\})$; furthermore, let us write $v \in V_i$ if $v \in V$ and v belongs to at least one edge of $\mathcal{H}'_i$; let us set $V_0 = V - (V_1 \cup V_2)$, $D = V - C$, and $C_i = C \cap V_i$, $D_i = D \cap V_i$ for $i = 0, 1, 2$. By assumption, V_1, V_2 are disjoint and C_0, D_0 are nonempty.

Now consider a tour T through $\mathcal{H}$, whose incidence vector x satisfies $\mathcal{H} \circ x = \mu(\mathcal{H})$. If this tour has the form $\ldots ab \ldots cd \ldots$ such that

- $a \in C$, $b \in D$, $c \in C$, $d \in D$,
- $a \notin C_1$ or $b \notin D_1$,
- $c \notin C_1$ or $d \notin D_1$,

then replace the pair of edges ab, cd with the pair of edges ac, bd. It is a routine matter to verify that the incidence vector y of the resulting tour satisfies $\delta_y(S) \leq \delta_x(S)$ for all edges S of $\mathcal{H}_1$. Similarly, if the tour has the form $\ldots ab \ldots cd \ldots$ with

- $a \in D$, $b \in C$, $c \in D$, $d \in C$,
- $a \notin D_1$ or $b \notin C_1$,
- $c \notin D_1$ or $d \notin C_1$,

then replace the pair of edges ab, cd with the pair of edges ac, bd. Again, it is a routine matter to verify that the incidence vector y of the resulting tour satisfies $\delta_y(S) \leq \delta_x(S)$ for all edges S of $\mathcal{H}_1$. Repeating these transformations as many times as possible, we eventually obtain a tour T_1 through V, whose incidence vector x_1 satisfies

$$\mathcal{H}'_1 \circ x_1 \leq \mathcal{H}'_1 \circ x \tag{7.1}$$

and

$$x_1(C,D) \leq x(C_1,D_1) + 2. \tag{7.2}$$

The same procedure with C_2, D_2 in place of C_1, D_1 yields a tour T_2 through V, whose incidence vector x_2 satisfies

$$\mathcal{H}'_2 \circ x_2 \leq \mathcal{H}'_2 \circ x \tag{7.3}$$

and

$$x_2(C,D) \leq x(C_2,D_2) + 2. \tag{7.4}$$

If $x_1(C,D) + x_2(C,D) \leq x(C,D) + 2$ then (7.1) and (7.3) guarantee that

$$\begin{aligned} \mathcal{H}_1 \circ x_1 + \mathcal{H}_2 \circ x_2 &= (\mathcal{H}'_1 \circ x_1 + x_1(C,D) - 2) + (\mathcal{H}'_2 \circ x_2 + x_2(C,D) - 2) \\ &\leq \mathcal{H}'_1 \circ x + \mathcal{H}'_2 \circ x + x(C,D) - 2 = \mu(\mathcal{H}), \end{aligned}$$

and the desired conclusion follows. Hence we may assume that

$$x_1(C,D) + x_2(C,D) > x(C,D) + 2.$$

Now, since

$$x_1(C,D) + x_2(C,D) \leq (x(C_1,D_1) + 2) + (x(C_2,D_2) + 2) \leq x(C,D) + 4$$

(the first inequality follows from (7.2) and (7.4); the second inequality is trivial) and since $x_1(C,D)$, $x_2(C,D)$, $x(C,D)$ are all even, we must have

$$\begin{aligned} x_1(C,D) &= x(C_1,D_1) + 2, & (7.5) \\ x_2(C,D) &= x(C_2,D_2) + 2, & (7.6) \\ x(C,D) &= x(C_1,D_1) + x(C_2,D_2). & (7.7) \end{aligned}$$

Let us say that an edge of T is of *type 1* if one of its endpoints belongs to C_1 and its other endpoint belongs to D_1; let us say that an edge of T is of *type 2* if one of its endpoints belongs to C_2 and its other endpoint belongs to D_2. By (7.7), each edge of T that crosses the boundary of C is either of type 1 or of type 2; by (7.5),

$$\text{the number of edges of type 1 is even;} \tag{7.8}$$

by (7.6),

$$\text{the number of edges of type 2 is even;} \tag{7.9}$$

by (7.1), (7.3), (7.5), (7.6), (7.7), we have

$$\mathcal{H}_1 \circ x_1 + \mathcal{H}_2 \circ x_2 \leq \mu(\mathcal{H}) + 2. \tag{7.10}$$

Let us choose once and for all one of the two cyclic orientations of T. We shall say that an edge of type 1 or type 2 is *outward* (with respect to our fixed orientation of T) if its endpoint in C precedes its endpoint in D; otherwise we shall say that the edge is *inward*. Now consider the cyclic sequence S of symbols $+$ and $-$ that arises from the cyclic sequence of edges of type 1 on T by writing a $+$ for each outward edge and writing a $-$ for each inward edge. If

($*$) S contains two consecutive identical symbols

then $S = aaS'$ for some S', whose length is even by virtue of (7.8). The block-substitution rules

$$++ \mapsto \Lambda, \quad -- \mapsto \Lambda, \quad +-+ \mapsto +, \quad -+- \mapsto -,$$

with Λ standing for the null sequence, transform S' into one of Λ, $+-$, $-+$, and so they transform the cyclic sequence S into Λ. Each of these block-substitutions translates into a replacement of a pair of edges ab, cd of type 1 with the pair of edges ac, bd; the entire sequence of these replacements transforms T into a tour T_2' through V, whose incidence vector x_2' satisfies

$$\mathcal{H}_2' \circ x_2' \leq \mathcal{H}_2' \circ x \quad \text{and} \quad x_2'(C, D) = x(C_2, D_2).$$

Now, by virtue of (7.1), (7.5), and (7.7),

$$\begin{aligned} \mathcal{H}_1 \circ x_1 + \mathcal{H}_2 \circ x_2' &= (\mathcal{H}_1' \circ x_1 + x_1(C, D) - 2) + (\mathcal{H}_2' \circ x_2' + x_2'(C, D) - 2) \\ &\leq \mathcal{H}_1' \circ x + \mathcal{H}_2' \circ x + x(C, D) - 2 = \mu(\mathcal{H}), \end{aligned}$$

and the desired conclusion follows again. Hence we may assume that ($*$) fails; to put it differently, we may assume that

$$\text{in the cyclic sequence of edges of type 1 on } T\text{, outward and inward edges alternate.} \tag{7.11}$$

Similarly, we may assume that

$$\text{in the cyclic sequence of edges of type 2 on } T\text{, outward and inward edges alternate.} \tag{7.12}$$

(Note that (7.11) is a strengthening of (7.8) and that (7.12) is a strengthening of (7.9).)

In addition, we may also assume that

$$\text{there is an edge of type 2:} \tag{7.13}$$

else $x(C_2, D_2) = 0$ and $T_1 = T$, contradicting (7.5) and (7.7). Similarly, we may assume that

$$\text{there is an edge of type 1.} \tag{7.14}$$

Digressing for a while, consider an arbitrary tour T', whose vertex-set V is partitioned into pairwise disjoint sets $C_0, C_1, C_2, D_0, D_1, D_2$, and let us write $C = C_0 \cup C_1 \cup C_2$, $D = D_0 \cup D_1 \cup D_2$. Again, let us say that an edge of this tour is of type i if one of its endpoints belongs to C_i and its other endpoint belongs to D_i; let us assume that

(i) each edge of T' with one endpoint in C and the other endpoint in D
is of type 1 or type 2,

(ii) there are an edge of type 1 and an edge of type 2.

Again, let us say that an edge of type 1 or type 2 is *outward* (with respect to a fixed orientation of T') if its endpoint in C precedes its endpoint in D and let us say that the edge is *inward* otherwise; let us assume that

(iii) in the cyclic sequence of edges of type 1 on T', outward and inward edges alternate,

(iv) in the cyclic sequence of edges of type 2 on T', outward and inward edges alternate.

Removal of all the edges of type 1 from T' breaks T' into a family $\mathcal{P}_1$ of paths; assumption (iii) guarantees that each of these paths either begins and ends in C_1 or begins and ends in D_1. Similarly, removal of all the edges of type 2 from T' breaks T' into a family $\mathcal{P}_2$ of paths; assumption (iv) guarantees that each of these paths either begins and ends in C_2 or begins and ends in D_2. Let A_1^C denote the union of vertex-sets of all the paths in $\mathcal{P}_1$ that begin and end in C_1; let A_1^D denote the union of vertex-sets of all the paths in $\mathcal{P}_1$ that begin and end in D_1; let A_2^C denote the union of vertex-sets of all the paths in $\mathcal{P}_2$ that begin and end in C_2; let A_2^D denote the union of vertex-sets of all the paths in $\mathcal{P}_2$ that begin and end in D_2. We claim that

$$A_1^C \cup A_2^C = V \text{ or else } A_1^D \cup A_2^D = V. \tag{7.15}$$

To justify this claim, we shall use induction on the number of edges of type 1 and type 2. We may assume that some point u of V is outside $A_1^C \cup A_2^C$, and so it belongs to $A_1^D \cap A_2^D$; let ab be the first edge of type 1 or type 2 that we encounter proceeding from u along T', and let the b be the successor of a. Since $u \in A_1^D \cap A_2^D$, we have either $a \in D_1, b \in C_1$ or $a \in D_2, b \in C_2$; symmetry allows us to assume that $a \in D_1, b \in C_1$. Let cd be the first edge of type 1 or type 2 that we encounter proceeding from b along T', and let the d be the successor of c. Trivially, we have $c \in C, d \in D$, and so either $c \in C_1, d \in D_1$ or $c \in C_2, d \in D_2$; the latter case is excluded since $u \in A_2^D$; hence $c \in C_1, d \in D_1$. If ab and cd are the only two edges of type 1 then $A_1^D \cup A_2^D = V$ and we are done; else the tour T'' obtained from T' by transferring all the interior points of the segment $ab \ldots cd$ from C to D satisfies (i), (ii), (iii), (iv) in place of T'. The transformation of T' into T'' transfers all the interior points of the segment $ab \ldots cd$ from $A_1^C \cap A_2^D$ to $A_1^D \cap A_2^D$, but otherwise it leaves $A_1^C, A_2^C, A_1^D, A_2^D$ unchanged; in particular, with respect to T'', we still have $A_1^C \cup A_2^C \neq V$. Now the induction hypothesis guarantees that, with respect to T'', we have $A_1^D \cup A_2^D = V$; it follows that, with respect to T', we also have $A_1^D \cup A_2^D = V$.

By virtue of (7.7), (7.13), (7.14), (7.11), and (7.12), our original tour T satisfies (i), (ii), (iii) and (iv) in place of T'. Hence (7.15) allows us to distinguish between two cases.

Case 1: $A_1^C \cup A_2^C = V$.

Consider an arbitrary vertex e in D_0. By assumption of this case, e belongs to at least one A_i^C; symmetry allows us to assume that $i = 1$. It is a routine matter to verify that the sequence of operations transforming T into T_2 maintains the following invariant:

the current tour has the form $\ldots aPd \ldots$ *such that* $a \in D_1$, $d \in D_1$, *and*

P *is a path that begins and ends in* C_1, *contains no edges of type 1, and includes* e.

In particular, T_2 is a concatenation of paths P and Q such that

- P begins and ends in C_1, contains no edges of type 1, and includes e,
- Q begins and ends in D_1, and contains no edges of type 1.

Let b denote the first point of P, let c denote the last point of P, let d denote the first point of Q, let a denote the last point of Q, and let f denote the successor of e on T_2. Now T_2 has the form

$$\ldots ab \ldots ef \ldots cd \ldots;$$

let y denote the incidence vector of the tour obtained from T_2 by replacing the three edges ab, ef, cd with the three edges ae, df, bc. It is a routine matter to verify that $y(C, V-C) = x_2(C, V-C) - 2$ and that $y(S, V-S) \leq x_2(S, V-S)$ for all edges S of $\mathcal{H}_2'$ (since none of a, b, c, d, e belong to an edge of $\mathcal{H}_2'$); hence $\mathcal{H}_2 \circ y \leq \mathcal{H}_2 \circ x_2 - 2$, and the desired conclusion follows from (7.10).

Case 2: $A_1^D \cup A_2^D = V$.

Replacing C with D in each of $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}$, we find ourselves in Case 1. □

To establish the upper bound, $\mu(\mathcal{H}) \leq t - 1$, of Theorem 7.1, we first prove several easy lemmas of independent interest. Let $\mathcal{H}$ be a hypergraph $(V, \mathcal{F})$ and let D be a subset of V. By $\mathcal{H} \backslash D$, the hypergraph obtained from $\mathcal{H}$ by *deleting* D, we mean the hypergraph $(V - D, \{S - D : S \in \mathcal{F}\})$.

LEMMA 7.1 *$\mu(\mathcal{H}\backslash D) \le \mu(\mathcal{H})$ for all $\mathcal{H}$ and D.*

PROOF. Since $\mathcal{H}\backslash(D_1 \cup D_2) = (\mathcal{H}\backslash D_1)\backslash D_2$, we may restrict ourselves to the case where $D = \{v\}$ for some point v of V. Given any tour $\ldots uvw \ldots$ through V, skip v to obtain a tour $\ldots uw \ldots$ through $V - \{v\}$; with x standing for the incidence vector of the old tour and with y standing for the incidence vector of the new tour, observe that

$$y(S - \{v\}, (V - \{v\}) - (S - \{v\})) \le x(S, V - S)$$

for all subsets S of V; this inequality implies at once that $\mu(\mathcal{H}\backslash\{v\}) \le \mu(\mathcal{H})$. □

Every hypergraph $(V, \mathcal{F})$ defines an equivalence relation on V by putting two points of V in the same equivalence class if and only if no member of $\mathcal{F}$ includes precisely one of these two points; we refer to these equivalence classes as the *atoms* of $(V, \mathcal{F})$. By the *type* of a hypergraph $\mathcal{H}$, we mean the hypergraph that arises from $\mathcal{H}$ by deleting a subset of size $|A| - 1$ from each atom A of $\mathcal{H}$.

LEMMA 7.2 *If $\mathcal{G}$ is the type of $\mathcal{H}$ then $\mu(\mathcal{G}) = \mu(\mathcal{H})$.*

PROOF. Repeated applications of Lemma 7.1 show that $\mu(\mathcal{G}) \le \mu(\mathcal{H})$. To see that $\mu(\mathcal{H}) \le \mu(\mathcal{G})$, consider an arbitrary tour $u_1, u_2, \ldots, u_m, u_1$ through $\mathcal{G}$ and let x denote the incidence vector of this tour. For each i, let P_i denote any path through the atom of $\mathcal{H}$ that contains u_i. The concatenation of $P_1, P_2, \ldots, P_m$ yields a tour through $\mathcal{H}$, whose incidence vector y satisfies $\mathcal{H} \circ y = \mathcal{G} \circ x$. □

By an *optimal* tour through a hypergraph $\mathcal{H}$, we shall mean any tour through $\mathcal{H}$, whose incidence vector x satisfies $\mathcal{H} \circ x = \mu(\mathcal{H})$.

LEMMA 7.3 *For every hypergraph $\mathcal{H}$ and for every nonempty subset B of an atom of $\mathcal{H}$ there is an optimal tour through $\mathcal{H}$, whose incidence vector x satisfies $\delta_x(B) = 0$.*

PROOF. Let A be the atom of $\mathcal{H}$ that contains B; let P be the concatenation of an arbitrary path through B and an arbitrary path through $A - B$. Let $\mathcal{G}$ be a hypergraph obtained from $\mathcal{H}$ by deleting a set of $|A| - 1$ points in A; consider an optimal tour through $\mathcal{G}$ and let x denote the incidence vector of this tour. Substituting P for the single point of A in this tour, we obtain a tour through $\mathcal{H}$, whose incidence vector y satisfies $\mathcal{H} \circ y = \mathcal{G} \circ x$; by Lemma 7.2, $\mathcal{G} \circ x = \mu(\mathcal{H})$. □

By an *end* of a clique-tree-like hypergraph $\mathcal{H}$, we shall mean any hypergraph $\mathcal{H}'$ such that
- $\mathcal{H}'$ is a comb or a flipped comb, and
- $\mathcal{H}$ arises from $\mathcal{H}'$ and some clique-tree like hypergraph $\mathcal{H}''$ by gluing along their common tooth.

LEMMA 7.4 *Let $\mathcal{H}$ be a clique-tree-like hypergraph. If $\mathcal{H}$ is neither a comb nor a flipped comb then $\mathcal{H}$ has at least two ends.*

PROOF. By induction on the number of edges of $\mathcal{H}$. By definition, $\mathcal{H}$ arises from clique-tree-like hypergraphs $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ by gluing along their common tooth T. We propose to find hypergraphs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$ such that each $\mathcal{G}_i$ is an end of $\mathcal{H}$ and all its edges come from $\mathcal{H}_i$. If $\mathcal{H}_i$ is a comb or a flipped comb then we may set $\mathcal{G}_i = \mathcal{H}_i$; else the induction hypothesis guarantees that $\mathcal{H}_i$ has distinct ends, say, $\mathcal{G}_{i1}$ and $\mathcal{G}_{i2}$. If at least one $\mathcal{G}_{ij}$ does not include T in its edge-set then this $\mathcal{G}_{ij}$ is an end of $\mathcal{H}$; else both $\mathcal{G}_{i1}$ and $\mathcal{G}_{i2}$ are ends of $\mathcal{H}$. $\square$

PROOF OF THEOREM 7.1. Let us write $\mathcal{H} = (V, \mathcal{F})$ and proceed by induction on the number of edges of $\mathcal{H}$. In Section 5, we observed that $\mu(\mathcal{H}) = t - 1$ whenever $\mathcal{H}$ is a comb with t teeth; since $\mu(\mathcal{H})$ is invariant under a substitution of $V - S$ for any S in $\mathcal{F}$, it follows that $\mu(\mathcal{H}) = t - 1$ whenever $\mathcal{H}$ is a flipped comb with t teeth; hence Lemma 7.4 allows us to assume that $\mathcal{H}$ arises from a comb or a flipped comb $\mathcal{H}_1$ and a clique-tree-like hypergraph $\mathcal{H}_2$ by gluing along their common tooth T. With t_i standing for the number of teeth of $\mathcal{H}_i$, we have $\mu(\mathcal{H}_1) = t_1 - 1$ and, by the induction hypothesis, $\mu(\mathcal{H}_2) = t_2 - 1$; trivially, $t = t_1 + t_2 - 1$.

Theorem 7.2 guarantees that $\mu(\mathcal{H}) \geq t - 1$.

To show that $\mu(\mathcal{H}) \leq t - 1$, let H denote the handle of $\mathcal{H}_1$ and enumerate the teeth of $\mathcal{H}_1$ other than T as $T_1, T_2, \ldots, T_{2k}$; note that $T_1, T_2, \ldots, T_{2k}$ are pairwise disjoint whether $\mathcal{H}_1$ is a comb or a flipped comb; set $B = T_1 \cup T_2 \cup \ldots \cup T_{2k} \cup H$. It is easy to construct a path P through B that
- starts in $T_1 - H$ and ends in $T_{2k} - H$,
- crosses the boudary of T_1 precisely once,
- crosses the boudary of each T_i with $1 < i < 2k$ precisely twice,
- crosses the boudary of T_{2k} precisely once,
- crosses the boundary of H precisely $2k$ times,
- crosses from $H \cap T$ to $H - T$ precisely twice.

Now set $T' = T - H$ if $\mathcal{H}_1$ is a comb and $T' = T \cup H$ if $\mathcal{H}_1$ is a flipped comb; in either case, let $\mathcal{H}'$ denote the hypergraph obtained from $\mathcal{H}_2$ by substituting T' for T. By Lemma 7.3, some optimal tour through $\mathcal{H}'$ is the concatenation of a path Q through $V - B$ and a path R through B. Substituting P for R, we obtain another optimal tour through $\mathcal{H}'$. The incidence vector x of this new tour satisfies $\delta_x(T_i) = 0$ for all i, $\delta_x(H) = 2k - 2$, and $\delta_x(T) = \delta_x(T') + 2$. Hence $\mathcal{H} \circ x = \mu(\mathcal{H}') + 2k$ and the desired conclusion follows since $\mu(\mathcal{H}') = \mu(\mathcal{H}_2)$ by Lemma 7.2. $\square$

Let us close this section with a brief discussion of clique tree inequalities and their relationship to clique-tree-like inequalities.

The *intersection graph* of a hypergraph $(V, \mathcal{F})$ is the graph with vertex-set $\mathcal{F}$, where two vertices are adjacent if and only if they have a nonempty intersection. A *clique tree* is any hypergraph $\mathcal{H}$ such that

- the intersection graph of $\mathcal{H}$ is a tree

and such that the edge-set of $\mathcal{H}$ can be partitioned into a set of "handles" and a set of "teeth" with the following properties:

- there is at least one handle,
- the handles are pairwise disjoint,
- the teeth are pairwise disjoint,
- the number of teeth that each handle intersects is odd and at least three,
- each tooth includes a point that belongs to no handle.

It is easy to show that a clique-tree-like hypergraph $\mathcal{H}$ is a clique tree if and only if

- each vertex of $\mathcal{H}$ belongs to at most two edges.

In the reverse direction, it is easy to show that a clique tree $\mathcal{H}$ is a clique-tree-like hypergraph if and only if $\mathcal{H}$ has at least one of the following two properties:

- some vertex of $\mathcal{H}$ belongs to no edge of $\mathcal{H}$,
- $\mathcal{H}$ is a comb.

A *clique tree inequality* is any inequality

$$\mathcal{H} \circ x \geq t - 1$$

such that $\mathcal{H}$ is a clique tree with t teeth. Grötschel and Pulleyblank [16] proved (among many other things) that every clique tree inequality is satisfied by all incidence vectors x of tours. As we have just observed, this result is not a corollary of our Theorem 7.2. (Nevertheless, it can be derived from Theorem 7.2 with just a little additional effort.)

8 FINDING CUTS: TIGHTENING

If the mechanism that drives a solution of an LP relaxation of a TSP instance to the minimum level of its objective function is likened to the force of gravity that drives a stream of water to the sea level, then each new cut added to the relaxation may be likened to a new obstacle in the way of the stream. The stream may change its course just slightly to get around the new obstacle and then go on its way; optimal solutions x of successive relaxations often exhibit an exasperating tendency to sidestep each new cut in a similar way. Fortunately, we may fight back with the same weapons and respond to each slight adjustment of x with a slight adjustment of the cut: in the metaphor, we may extend previously built dams rather than blowing them up and building new ones from scratch. Given x, we may scan our cutpool and try to adjust each of its hypergraphs $\mathcal{H}$, so that resulting hypergraph $\mathcal{H}'$ satisfies $\mathcal{H}' \circ x < \mu(\mathcal{H}')$.

LEMMA 8.1 *Let $\mathcal{H}$ be a hypergraph, let S be an edge of $\mathcal{H}$, and let v be a vertex in S; assume that the atom of $\mathcal{H}$ that contains v has size at least two; in the edge-set of $\mathcal{H}$, replace S with $S - \{v\}$ and call the resulting hypergraph $\mathcal{H}'$. Then $\mu(\mathcal{H}') \geq \mu(\mathcal{H})$.*

PROOF. Let $\mathcal{G}$ be the type of $\mathcal{H}$ and let $\mathcal{G}'$ be the type of $\mathcal{H}'$. By assumption, either $\mathcal{G} = \mathcal{G}'$ or $\mathcal{G}$ arises from $\mathcal{G}'$ by deleting a single vertex. Hence the desired conclusion follows from Lemma 7.2 and Lemma 7.1. □

Here is a function `shrink`$(\mathcal{H}, S)$ that, given any hypergraph $\mathcal{H}$ and its edge S, replaces S in the edge-set of $\mathcal{H}$ with a subset S' of S and returns the resulting hypergraph $\mathcal{H}'$; in a greedy way, this function attempts to minimize $\delta_x(S')$ subject to constraints $\mu(\mathcal{H}') \geq \mu(\mathcal{H})$ and $S' \subseteq S$.

```
initialize an empty queue;
for all $v$ in $S$
do $f(v) = x(\{v\}, S)$;
   if $f(v) \leq 1$ then place $v$ in the queue;
end
while the queue is nonempty
do     $v=$ the first vertex in the queue;
       remove $v$ from the queue;
       if the atom containing $v$ has size at least two
       then in the edge-set of $\mathcal{H}$, replace $S$ with $S - \{v\}$;
            for all neighbors $w$ of $v$ such that $w \in S$
            do $t = f(w)$;
               decrement $f(w)$ by $x(\{v\}, \{w\})$
               if $f(w) \leq 1 < t$ then place $w$ in the queue;
            end
       end
end
return $\mathcal{H}$;
```

Function **grow**$(\mathcal{H}, S)$ is a mirror image of **shrink**$(\mathcal{H}, S)$:

```
initialize an empty queue;
for all $v$ outside $S$
do $f(v) = x(\{v\}, S)$;
   if $f(v) \geq 1$ then place $v$ in the queue;
end
while the queue is nonempty
do     $v=$ the first vertex in the queue;
       remove $v$ from the queue;
       if the atom containing $v$ has size at least two
       then in the edge-set of $\mathcal{H}$, replace $S$ with $S \cup \{v\}$;
            for all neighbors $w$ of $v$ such that $w \notin S$
            do $t = f(w)$;
               increment $f(w)$ by $x(\{v\}, \{w\})$
               if $t < 1 \leq f(w)$ then place $w$ in the queue;
            end
       end
end
return $\mathcal{H}$;
```

Function **tighten**$(\mathcal{H}, S)$ makes S wane and wax as long as its $\delta(S)$ keeps getting smaller; we insist on waning first and waning last to improve our chances of ending up with a relatively small S, and so easing the burden of our LP solver.

```
$\mathcal{H}$=shrink($\mathcal{H}, S$);
do $t = \delta(S)$;
    $\mathcal{H}$=grow($\mathcal{H}, S$);
    $\mathcal{H}$=shrink($\mathcal{H}, S$);
while $\delta(S) < t$
```

We have also implemented more sophisticated versions of **tighten** that tighten edges by solving max-flow min-cut problems; however, the greedy versions described here turned out to perform quite adequately.

Given a hypergraph $\mathcal{H}$ in our cutpool, we arrange its m edges in a cyclic sequence; then we go through this cyclic sequence edge by edge, applying **tighten**$(\mathcal{H}, S)$ to each term S; we stop only when the last m calls of **tighten** together made the value of $\mathcal{H} \circ x$ drop by an insignificant amount or not at all. We refrain from struggle for lost causes: if the initial value of $\mathcal{H} \circ x - \mu(\mathcal{H})$ is unreasonably large (say, greater than two) that we do not even attempt to tighten $\mathcal{H}$.

ACKNOWLEDMENTS

In the summer of 1988, Bernhard Korte took three of us into his Institut für Diskrete Mathematik, where we could get our project off the ground under ideal working conditions; his moral and material support has continued through the subsequent years. We want to thank him for his unswerving faith in us and for his hospitality.

We also want to thank the National Science Foundation for its SGER grant CCR–9305589 that helped us to free the time required for writing up this preliminary report.

References

[1] R. E. Bland and D. F. Shallcross, "Large traveling salesman problems arising from experiments in X-ray crystallography: a preliminary report on computation", *Oper. Res. Lett.* **8** (1989), 125–128.

[2] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms", *J.Comput.Syst.Sci.* **13** (1976), 335–379.

[3] O. Borůvka, "On a certain minimal problem" (in Czech), *Práce Moravské Přírodovědecké Společnosti* **3** (1926), 37–58.

[4] V. Chvátal, "Edmonds polytopes and weakly hamiltonian graphs", *Math. Programming* **5** (1973), 29–40.

[5] W. Cook and M.Hartmann, "On the complexity of branch and cut methods for the traveling salesman problem", in: *Polyhedral combinatorics* (W. Cook and P.D.Seymour, eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science **1**, Amer. Math. Soc., 1990, pp. 75–81.

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

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

[8] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs", *Bull. Amer. Math. Soc.* **64** (1958), 275–278.

[9] R. E. Gomory, "Solving linear programs in integers", in: *Combinatorial Analysis* (R. E. Bellman and M. Hall, Jr., eds.), Proc. Symp. Appl. Math. **X** (1960), 211–216.

[10] R. E. Gomory, "An algorithm for integer solutions to linear programs", in: *Recent Advances in Mathematical Programming* (R. L. Graves and P. Wolfe, eds.), McGraw-Hill, New York, 1963, pp. 269–302.

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

[12] M. Grötschel and O. Holland, "Solution of large-scale symmetric travelling salesman problems", *Math. Programming* **51** (1991) 141–202.

[13] M. Grötschel, M. Jünger, and G. Reinelt, "A cutting plane algorithm for the linear ordering problem", *Operations Research* **32** (1984) 1195–1220.

[14] M. Grötschel and M. W. Padberg, "On the symmetric travelling salesman problem I: Inequalities", *Math. Programming* **16** (1979), 265–280.

[15] M. Grötschel and M. W. Padberg, "On the symmetric travelling salesman problem II: Lifting theorems and facets", *Math. Programming* **16** (1979), 281-302.

[16] M. Grötschel and W. Pulleyblank, "Clique tree inequalities and the symmetric travelling salesman problem", *Math. Oper. Res.* **11** (1986), 537–569.

[17] D. S. Johnson and C. H. Papadimitriou, "Computational complexity", in: *The Traveling Salesman Problem* (E. L. Lawler et al., eds.), Wiley, Chichester, 1995, pp. 37–85.

[18] R. M. Karp, "Reducibility among combinatorial problems", in: *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, eds.), Plenum Press, 1972, pp.85–103.

[19] S. Lin and B. W. Kerninghan, "An effective heuristic algorithm for the traveling-salesman problem", *Oper. Res.* **21** (1973), 498–516.

[20] P. Miliotis, "Using cutting planes to solve the symmetric traveling salesman problem", *Mathematical Programming* **10** (1976) 367-378.

[21] M. W. Padberg and M. Grötschel, "Polyhedral computations", in: *The Traveling Salesman Problem* (E. L. Lawler et al., eds.), Wiley, Chichester, 1995, pp.307–360.

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

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

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

[25] M. W. Padberg and G. Rinaldi, "Facet identification for the symmetric traveling salesman polytope", *Math. Programming* **47** (1990), 219–257.

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

[27] G. Reinelt, "TSPLIB – A traveling salesman problem library", *ORSA J. Computing* **3** (1991), 376–384.

[28] G. Reinelt, "TSPLIB – Version 1.2", *Report No.330, Schwerpunktprogramm der Deutschen Forschungsgemeinschaft*, Universität Augsburg, 1991.

[29] A. Tucker, "A structure theorem for the consecutive 1's property", *J. Combin. Theory Ser. B* **12** (1972), 153–162.