Chapter 10

Lower Bounds

When solving TSPs in practice, the main interest, of course, lies in computing good feasible tours. But, in addition, one would like to have some guarantee on the quality of the solutions found. Such guarantees can, except for weak a-priori guarantees for some heuristics, only be given if a lower bound for the length of a shortest possible tour is known.
In general, lower bounds are obtained by solving relaxations of the original problem in the sense that one optimizes over some set containing all feasible solutions of the original problem as a (proper) subset. Then (for a minimization problem) the optimal solution of the relaxed problem gives a valid lower bound for the value of the optimal solution of the original problem. Different relaxations provide different lower bounds. The main goal is to find relaxation problems for the TSP that can be solved efficiently and which are as tight as possible under this constraint.
For the purposes of this study, we are mainly interested in lower bounds which can be computed fast enough to only slightly increase the overall computation effort and running time. These lower bounds are not meant to give a very good estimate of the achievable optimum (say within a few percent) but to give indications on the quality of tours found by fast heuristics. This is usually sufficient for practitioners to get an impression of the performance of a heuristic on a particular problem.
But, to really evaluate a heuristic one should spend more time for computing better lower bounds. We will also address this question and indicate how CPU time can be saved.

10.1 Bounds from Linear Programming

Employing linear programming to determine lower bounds for the traveling salesman problem is not a major topic of this monograph. We will return to approaches for solving the TSP to optimality using linear programming bounds in Chapter 12. For the purposes of this chapter we need the theoretical framework provided by linear programming. The (combinatorial) bounds to be discussed in the following section can be compared with respect to related linear programming relaxations.
Consider the **traveling salesman polytope** $P_T = \text{conv}\{\chi^F \in \{0,1\}^{\binom{n}{2}} \mid \chi^F \text{ is the incidence vector of tour } F \text{ in } K_n = (V_n, E_n)\}$, i.e., the convex hull of the incidence vectors of all tours in the complete graph. For a given vector c of edge lengths the optimal solution of the corresponding traveling salesman problem instance is obtained

G. Reinelt: The Traveling Salesman, LNCS 840, pp. 161-186, 1994.

by solving the linear programming problem $\min\{c^T x \mid x \in P_T\}$. The question of solving this problem will be addressed in Chapter 12.
Now let P be a polytope (or polyhedron) such that $P_T \subseteq P$. A solution of the problem $\min\{c^T x \mid x \in P\}$ yields a lower bound on the minimal tour length. Depending on how well P approximates P_T this lower bound may come close to the optimal tour length. Two relaxations are of particular interest for the following.

10.1.1 The 2-Matching Relaxation

A **perfect 2-matching** in a graph $G = (V, E)$ is a set of edges such that every node of V is incident to exactly two of these edges. Every tour is therefore a perfect 2-matching, but since also a collection of subtours is a perfect 2-matching we only have a relaxation. The following is an integer linear programming formulation of the 2-matching problem.

$$\begin{aligned} \min \sum_{ij \in E_n} c_{ij} x_{ij} & \\ x(\delta(i)) &= 2, \qquad \text{for all } i \in V_n, \\ x_{ij} &\in \{0,1\}, \text{ for all } ij \in E_n. \end{aligned}$$

This problem can be solved in polynomial time (EDMONDS & JOHNSON (1973)). Implementation of this algorithm is nontrivial, its worst case running time is $O(n^3)$. An efficient implementation is discussed in PEKNY & MILLER (1994).
If we replace the requirement "$x_{ij} \in \{0,1\}$" by "$0 \leq x_{ij} \leq 1$" we obtain the **fractional 2-matching relaxation** of the traveling salesman problem.

10.1.2 The Subtour Elimination Relaxation

In the 2-matching problem only the degree constraints are taken into account. Short cycles are not forbidden. If we include conditions to eliminate such subtours we obtain the following integer linear program.

$$\begin{aligned} \min \sum_{ij \in E_n} c_{ij} x_{ij} & \\ \sum_{j=1}^{n} x_{ij} &= 2, \qquad \text{for all } i \in V_n, \\ x(E(S)) &\leq |S| - 1, \quad \text{for all } S \subseteq V_n, 2 \leq |S| \leq \lfloor \frac{n}{2} \rfloor, \\ x_{ij} &\in \{0,1\}, \qquad \text{for all } ij \in E_n. \end{aligned}$$

Note that this formulation is equivalent to the one given in section 2.3. Feasible solutions of this problem are exactly the incidence vectors of tours. Therefore solving this problem is $\mathcal{NP}$-hard. Relaxing the integrality stipulations and observing that the constraints for the 2-element sets S yield upper bounds for the variables we obtain the **subtour**

elimination relaxation of the TSP.

$$\begin{aligned} \min & \sum_{ij \in E_n} c_{ij} x_{ij} \\ \sum_{j=1}^{n} x_{ij} &= 2, && \text{for all } i \in V_n, \\ x(E(S)) &\leq |S| - 1, && \text{for all } S \subseteq V_n, 2 \leq |S| \leq \lfloor \frac{n}{2} \rfloor, \\ x_{ij} &\geq 0, && \text{for all } ij \in E_n. \end{aligned}$$

An equivalent formulation of this linear programming problem (Grötschel & Padberg (1985)) is the following.

$$\begin{aligned} \min & \sum_{ij \in E_n} c_{ij} x_{ij} \\ \sum_{j=1}^{n} x_{ij} &= 2, && \text{for all } i \in V_n, \\ x(\delta(W)) &\geq 2, && \text{for all } W \subseteq V_n, 2 \leq |W| \leq \lfloor \frac{n}{2} \rfloor, \\ x_{ij} &\geq 0, && \text{for all } ij \in E_n. \end{aligned}$$

Here subtour elimination constraints are given in their cut version. This corresponds to requiring that every (nonempty) cut in K_n contains at least two tour edges.
The subtour elimination bound can be determined in polynomial time using the ellipsoid method (Grötschel, Lovász & Schrijver (1988)) based on a polynomial time separation algorithm for subtour elimination constraints (Padberg & Grötschel (1985)). So far, no "nice" algorithm, i.e., an algorithm which does not explicitly need an LP solver as subroutine, for solving this problem in polynomial time is known. Some interesting properties of this relaxation are studied in Boyd & Pulleyblank (1990).

10.2 Simple Lower Bounds

We start the discussion of lower bounds for the TSP by considering several fairly simple bounds. These bounds are "combinatorial" in the sense that they are derived directly as obvious relaxations of the definition of tours.

10.2.1 The 1-Tree Bound

The **1-tree bound** for the TSP is based on the following observation. If we select some node of the problem, say node 1, then a Hamiltonian tour consists of a special spanning tree (namely a path) on the remaining $n-1$ nodes plus two edges connecting node 1 to this spanning tree. Hence we obtain a relaxation of the TSP if we take as feasible solutions arbitrary spanning trees on the node set $V_n \setminus \{1\}$ plus two additional edges

incident to node 1. Of course, in the case of nonnegative edge lengths, the weight of the minimum spanning tree alone also provides a (weaker) lower bound.
Figure 10.1 displays a 1-tree for problem `pcb442`. In this 1-tree the special node is the node in the lower right corner. The corresponding lower bound is 46858. Recall that a shortest possible tour has length 50778.

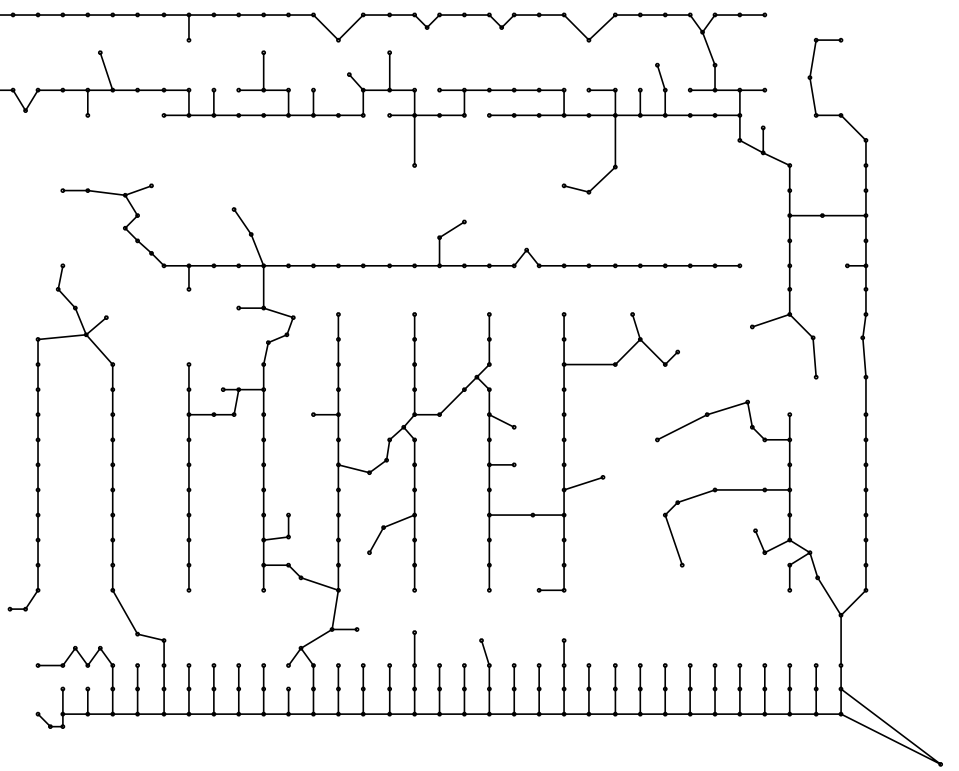

Figure 10.1 A 1-tree lower bound for `pcb442`

We use the following procedure to determine a 1-tree lower bound.

procedure simple_1tree

(1) Compute a minimum weight spanning tree T and let $c(T)$ be its weight.

(2) For every node i which is a leaf of this spanning tree compute the distance $d_2(i)$ to its second nearest neighbor (an edge to the nearest neighbor is already in T). This gives the lower bound $c(T) + d_2(i)$ on the minimal tour length.

(3) Take the best of the bounds computed in Step (2).

end of simple_1tree

Note, that we do not compute the best obtainable 1-tree. We just consider those nodes as special nodes which have degree 1 in the minimum spanning tree, and we take the best of these lower bounds. To compute the best 1-tree we have to compute n minimum spanning trees which is too time consuming.

In general, computing a minimum spanning tree in a complete graph on n nodes takes time $O(n^2)$, and finding the additional edge takes time $O(n)$ for every leaf. If we are not interested in computing several 1-trees we can compute a minimum spanning tree on the nodes $2, 3, \ldots, n$ in time $O(n^2)$ and add the two shortest edges incident to node 1 in time $O(n)$.

For geometric problems, we can do better by exploiting the Delaunay graph. Recall that the Delaunay graph is planar and therefore has $O(n)$ edges. Since the Delaunay graph contains a minimum spanning tree (for the complete graph) this spanning tree can be computed in the Delaunay graph in time $O(n \log n)$ using Kruskal's algorithm. Computing the various 1-trees for the leaves can be done with little additional time. For every node i we only have to compute the distances to nodes that are connected to i in the Delaunay graph by a path of length at most two. In the usual case these are only very few nodes.

The 1-tree computation constitutes a relaxation of the problem of finding a shortest Hamiltonian tour since we do not require every node to have degree 2. If a minimum 1-tree computed as above happens to satisfy this degree constraint, then it is an optimal tour. Unfortunately, this can never be expected for practical problems.

For other metrics we can employ the respective Delaunay graphs to speed up computations as well. With the help of the Delaunay graph we can determine the possible best 1-tree in time $O(n^2 \log n)$. Computing this 1-tree, however, turned out not to be worthwhile, because it only slightly (if at all) improves our simple 1-tree bound.

Frieze (1979) describes a tour construction heuristic based on 1-trees, that yields tours at most $2 - (k/n)$ times longer than an optimal tour in time $O(n^{3+k})$ for $1 \leq k \leq n-2$.

The 1-tree bound can be adapted in a fairly natural way to the asymmetric traveling salesman problem. For the ATSP, so-called **spanning 1-arborescences** are relaxations of directed Hamiltonian cycles. A 1-arborescence is an arc set with the property that every node has indegree at most 1 and a special node has indegree and outdegree equal to 1. The determination of minimum weight spanning 1-arborescences is more complicated than the determination of minimum spanning 1-trees, but can still be done very efficiently (Fischetti & Toth (1993).

10.2.2 The 2-Neighbor Bound

In a tour, each node is connected to exactly two other nodes. If we have a tour in which each node is connected to its two nearest neighbors then this tour must be optimal.

Of course, this can only be achieved in very rare cases. In general, this observation leads to the derivation of a further simple lower bound. If we compute for all nodes the distances to their two nearest neighbors, sum up all these distances and divide by 2 then this number (rounded up) gives a lower bound on the minimal tour length. We call the subgraph of the complete graph which is obtained by taking for each node the two edges to its two nearest neighbors a **2-neighbor configuration**. In this case we keep multiple edges, so that such a configuration consists of $2n$ edges.

Figure 10.2 visualizes the 2-neighbor lower bound for problem `pcb442`. It provides the lower bound 47304.

The 2-neighbor bound can be computed trivially in time $O(n^2)$. For geometric instances we can do better by exploiting the Delaunay graph. Having computed the Delaunay

graph in time $O(n \log n)$ we only have to compute for every node the distances to those nodes that are connected to it by a path of length at most two. The two nearest neighbors are among these nodes.

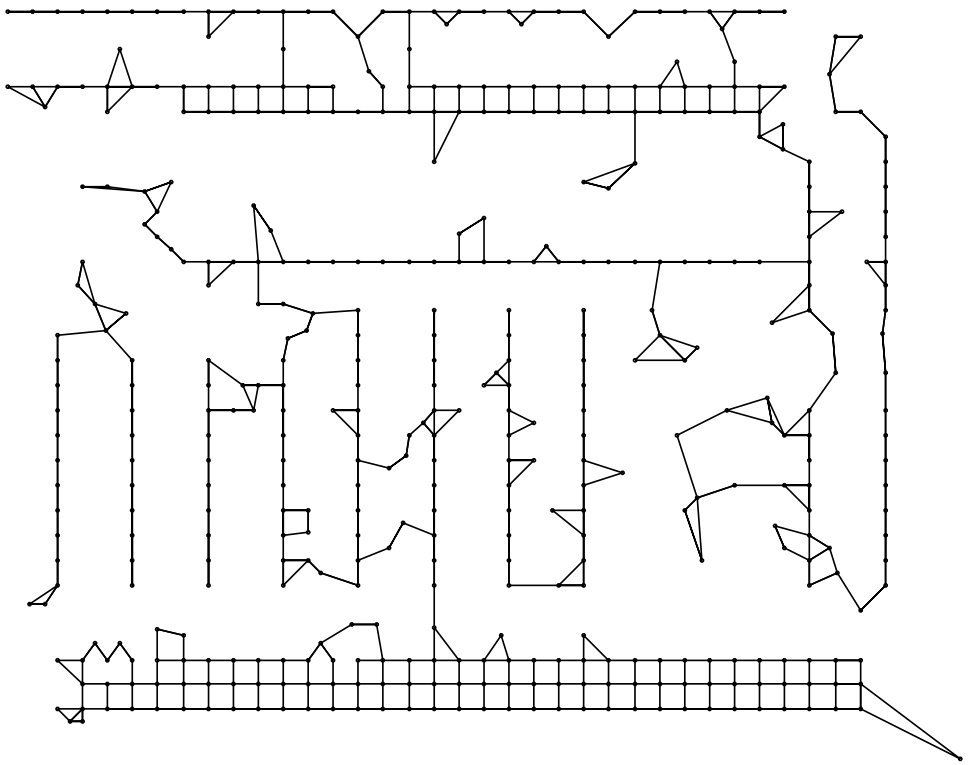

Figure 10.2 The 2-neighbor bound for `pcb442`

The quality of the bounds is, of course, highly problem dependent. The 2-neighbor configurations can give very poor bounds if there are clusters of points. Some such examples can be found in the computational results (e.g., `d198`, `p654`, `u1060` or `rl5934`). A large gap between 1-tree and 2-neighbor bound might indicate that the problem is well suited for decomposition algorithms (described in Chapter 8) and also that the nearest neighbor candidate set is not sufficient for finding good tours.

10.2.3 The Assignment Relaxation

Another example of an relaxation that is often discussed in the context of TSP relaxations is the assignment relaxation. It is usually employed for asymmetric problem instances, but can also be formulated for the undirected problem.
An **assignment** for $V_n = \{1, 2, \ldots, n\}$ is a collection S of ordered pairs of the form $S = \{(i, n_i) \mid i = 1, 2 \ldots, n\}$ such that every node occurs exactly once as the second component of a pair.
A tour gives a particular assignment as follows: we choose one of the two possible orientations of the tour and assign to every node its successor in the tour according to the chosen orientation. Such assignments have the additional property that for every pair (i, n_i) we have $i \neq n_i$. The cost of the assignment is exactly the length of the tour.

Conversely, we can associate with every assignment the undirected graph given by the edges $\{i, n_i\}$, $i = 1, 2, \ldots, n$. This graph can have loops or multiple edges. If we restrict ourselves to assignments not containing pairs (i, n_i) with $i = n_i$ then the associated graph consists only of multiple edges and cycles.

Therefore, the assigment problem constitutes a relaxation of the TSP. It can be solved in time $O(n^3)$ with several algorithmic approaches, e.g. with the Hungarian method (see Carpaneto & Toth (1983) for an efficient implementation).

If we do not allow loops, then the following is an integer linear programming formulation of the assignment relaxation for the TSP.

$$\begin{aligned} \min \sum_{ij \in E_n} c_{ij} x_{ij} & \\ x(\delta(i)) &= 2, \qquad \text{for all } i \in V_n, \\ x_{ij} &\in \{0, 1, 2\}, \text{ for all } ij \in E_n. \end{aligned}$$

The similarity to the 2-matching relaxation 10.1.1 is immediately seen. The important difference is that edge variables are allowed to have values greater than 1. Therefore, this relaxation is weaker than the 2-matching relaxation.

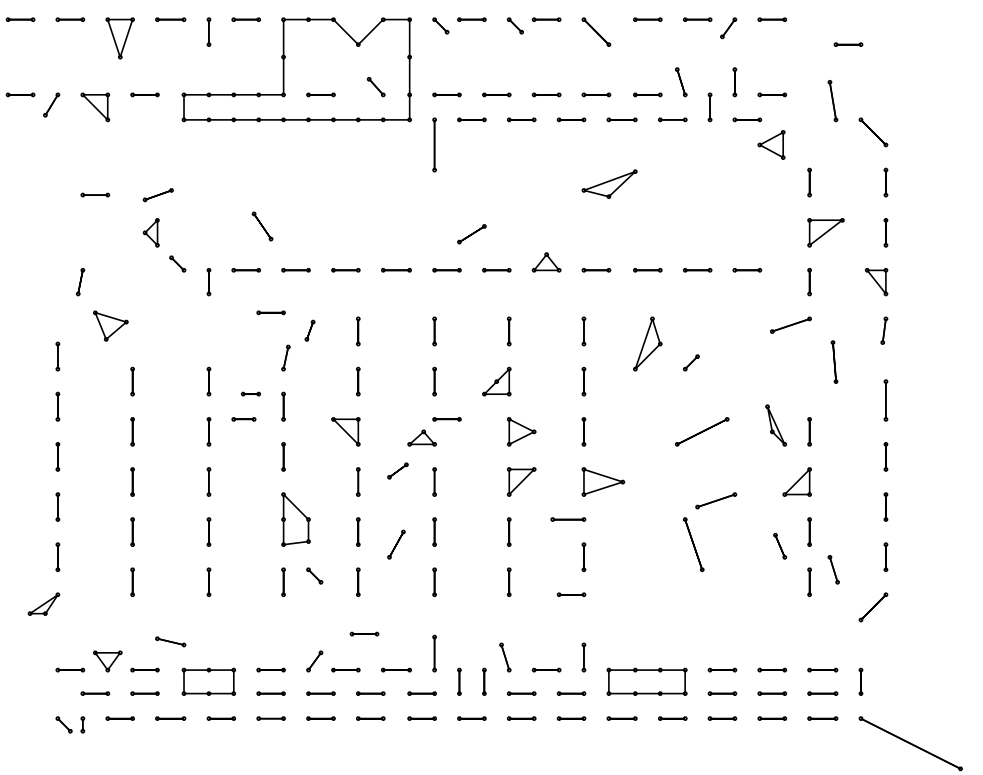

Figure 10.3 The assignment lower bound for `pcb442`

Figure 10.3 show the assignment lower bound for problem `pcb442` with value 46830. The optimal assignment does not even come close to a tour. It contains many multiple edges and only short cycles.

10.2.4 Geometric Bounds

The idea of the geometric lower bound described in the sequel originates from techniques applied to the Euclidean matching problem (JÜNGER & PULLEYBLANK (1993)).
The geometric structure of Euclidean TSPs yields a very simple illustrative lower bound for the TSP. We compute a system of circles around nodes and moats around sets of circles and moats. This is done in such a way that circles and moats do not overlap each other. Moreover, there has to be always at least one node inside and outside of each circle and moat.
Each node has to be contained in a tour and every moat has to be crossed at least two times (since there are nodes inside and outside of each moat). Hence twice the sum of the radii of all circles and the width of all moats gives a lower bound on the minimal tour length.
Figure 10.4 gives an illustration of such a system consisting of 7 circles and 5 moats.

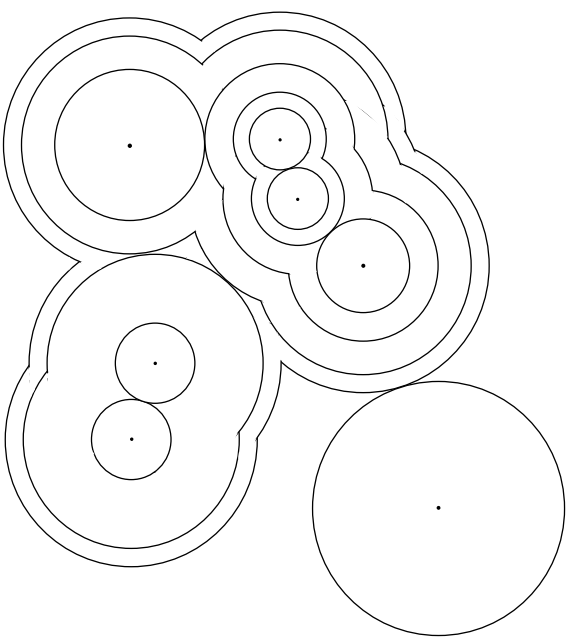

Figure 10.4 A system of circles and moats

Different systems of circles and moats are possible for a collection of points. One such geometric lower bound can be computed by extending Kruskal's algorithm with little additional computational effort. We give the procedure below.

procedure geometric_bound

(1) For the sake of simplicity we shall always speak about moats in the sequel (circles are just moats around a single node). Whenever a tree edge is selected during Kruskal's algorithm, it connects two components to form a new component. At this point a weight w will be assigned to the new component. This weight depends on the connecting edge and on the weights of the two participating components. Initially all components are just single nodes and their respective weights are 0. The lower bound lb is also set to 0.

(2) Let e be a tree edge with length c_e selected by Kruskal's algorithm. Let C_1 and C_2 be the two components to be connected to form the new component C. We set $lb := lb + 2 \cdot (c_e - w_{C_1} - w_{C_2})$ and $w_C := c_e/2$, where w_C, w_{C_1}, and w_{C_2} are the weights associated with the respective components.

(Note that the weights are not the moat widths. The widths of the moats around the components C_1 and C_2 used here implicitly are $\frac{1}{2}c_e - w_{C_1}$ and $\frac{1}{2}c_e - w_{C_2}$.)

(3) The value lb computed above gives a geometric lower bound based on circles and moats.

end of geometric_bound

The geometric lower bound as defined above can be obtained with (practically) no additional running time when implemented as in this procedure. Looking more closely at the way how this particular lower bound is computed, one realizes that it is exactly the sum of the length of a minimum spanning tree and the length of the final edge added by Kruskal's algorithm, i.e., the longest edge of the minimum spanning tree. Therefore, this specific computation is not restricted to geometric instances only. It applies to arbitrary TSPs.

The geometric bound seems to be rather weak. But we shall see below that the weak bounds obtained are only due to our simple scheme for determining the radii of the circles and the widths of the moats.

If we denote by z_i the radius of the circle around node i and by y_S the width of the moat around set S, $2 \le |S| \le n-1$, then the problem of finding the best bound can be formulated as a linear programming problem as follows.

$$
\begin{aligned}
& \max \quad 2\sum_{i=1}^{n} z_i + 2\sum_{S} y_S \\
\text{(CM)} \qquad & z_i + z_j + \sum_{\substack{S \\ i \in S, j \notin S}} y_S \le c_{ij}, \qquad \text{for all } ij \in E_n, \\
& z_i \ge 0, \qquad \text{for all } i \in V_n, \\
& y_S \ge 0, \qquad \text{for all } 2 \le |S| \le n-1.
\end{aligned}
$$

Dualizing this linear program we obtain

$$
\begin{aligned}
& \min \quad \sum_{ij \in E_n} c_{ij} x_{ij} \\
\text{(CM}_D\text{)} \qquad & \sum_{j=1}^{n} x_{ij} \ge 2, \qquad \text{for all } i \in V_n, \\
& \sum_{\substack{S \\ i \in S, j \notin S}} x_{ij} \ge 2, \qquad \text{for all } 2 \le |S| \le n-1, \\
& x_{ij} \ge 0, \qquad \text{for all } ij \in E_n.
\end{aligned}
$$

Note that the first group of inequalities corresponds to a set S of cardinality 1 in the second system.

It is known that every vertex x^* of the polytope defining (CM_D) is given as the unique solution of a system of $\binom{n}{2}$ equations of the form

$$\begin{aligned} x_{ij} &= 0, \quad \text{for all } ij \in F, \\ \sum_{i \in S, j \notin S} x_{ij} &= 2, \quad \text{for all } S \in \mathcal{B} \subseteq 2^{V_n}, |S| \geq 1, \end{aligned}$$

where $\mathcal{B}$ is a nested family, i.e., for every $S_i, S_j \in \mathcal{B}$ we have either $S_i \subset S_j$, $S_j \subset S_i$, or $S_i \cap S_j = \emptyset$, and where $F \subseteq E_n, |F| = \binom{n}{2} - |\mathcal{B}|$ (CORNUEJOLS, FONLUPT & NADDEF (1985), BOYD & PULLEYBLANK (1990)). Due to this condition we obtain $|\mathcal{B}| \leq 2n-1$. In our application we have $c_{ij} > 0$ for all $ij \in E_n$ and we are looking for the minimizer of (CM_D) (i.e., the maximal circles and moats lower bound). Let x^* be the minimizer. It is easy to see that for every $i \in V_n$ we must have $\sum_{j=1}^{n} x_{ij} = 2$. Suppose this is not the case for some $i \in V_n$ and let S be the minimal element (with respect to inclusion) of $\mathcal{B}$ containing node i. Then there is an edge ik with $x^*_{ik} > 0$ and $k \in S$ (otherwise $S \notin \mathcal{B}$). We can set $x^*_{ik} = \max\{0, 2 - \sum_{j \neq k} x^*_{ij}\}$ without violating any condition and obtain a new solution having strictly less objective function value.
This proves that, for a nonnegative objective function, the best circles and moats bound is equivalent to the subtour relaxation bound. Therefore, as noted above, this bound can be determined in polynomial time making use of the ellipsoid method. It would be interesting to design efficient heuristics providing good systems of circles and moats.

10.2.5 Computations

We have evaluated the above bounds for our set of sample problems. Except for the case of the assignment bound all bounds were computed exactly. Because of the running time $O(n^3)$ of assignment algorithms for complete graphs, the assignment bound was only computed for the subgraph consisting of the Delaunay graph and the 10 nearest neighbor subgraph. We tested several cases and always found that the assignment computed for this subgraph had the same value as the minimum weight assignment computed for the complete graph.
Table 10.5 displays the qualities of the computed bounds relative to the best known upper bounds. I.e., if c_L is a lower bound, we define its **quality** as $100 \cdot (c_L - c_U)/c_U$ where c_U is the length of the best known tour as given in Table 3.1. Best qualities in each row are marked.
The table gives a clear picture. The 1-tree and the related geometric bound perform best, providing on the average bounds about 10% below the optimal objective function value. The 2-neighbor and the assignment bounds are significantly worse. In some cases, however, the 2-neighbor bound is better than the tree bounds. Therefore, since it is quickly computed, it is worthwhile to be also used as a fast lower bounding procedure accompanying the tree bounds.
We end this section with an impression of the necessary CPU times. Figure 10.6 shows the running times for minimum spanning tree, the simple 1-tree and the 2-neighbor configuration bounding procedures. Running times are given without the necessary preprocessing times for computing the Delaunay graph. The figure shows that very little additional time is needed and that CPU times are well predictable.

Problem	MST	1-tree	Geometric	2-neighbor	Assignment
d198	25.61	18.16*	17.08	37.31	32.78
lin318	9.94	9.21	8.78*	18.91	35.17
fl417	14.27	11.98	10.37*	30.66	35.50
pcb442	8.70	7.72	7.82	6.84*	7.78
u574	13.08	12.14*	12.25	17.03	20.97
p654	14.97	12.99	11.58*	29.11	32.14
rat783	7.73	7.44*	7.47	9.33	15.47
pr1002	13.46	12.82	12.66*	15.07	17.23
u1060	12.78	11.72*	11.96	16.49	18.15
pcb1173	9.63	9.20*	9.25	8.58	10.18
d1291	7.62	5.08*	5.11	16.98	19.83
rl1323	11.18	10.82	10.43*	18.93	23.32
fl1400	15.20	12.53*	12.90	31.01	39.40
u1432	4.57	4.38	4.39	3.39*	3.42
fl1577	12.62	12.00	10.67*	23.34	24.77
d1655	8.99	6.62*	6.64	12.15	13.73
vm1748	12.46	12.16*	12.20	14.40	17.01
rl1889	12.09	11.85	11.83*	18.71	22.87
u2152	4.16	4.00*	4.00*	7.80	12.10
pr2392	9.46	9.35	9.33*	10.66	15.49
pcb3038	7.55	7.42	7.42	6.69*	8.00
fl3795	12.18	10.33*	10.86	21.39	19.23
fnl4461	7.73	7.65*	7.66	6.83	9.97
rl5934	7.24	7.17	7.09*	13.64	17.76
Average	10.97	9.78	9.57	16.47	19.68

Table 10.5 Quality of lower bounding procedures

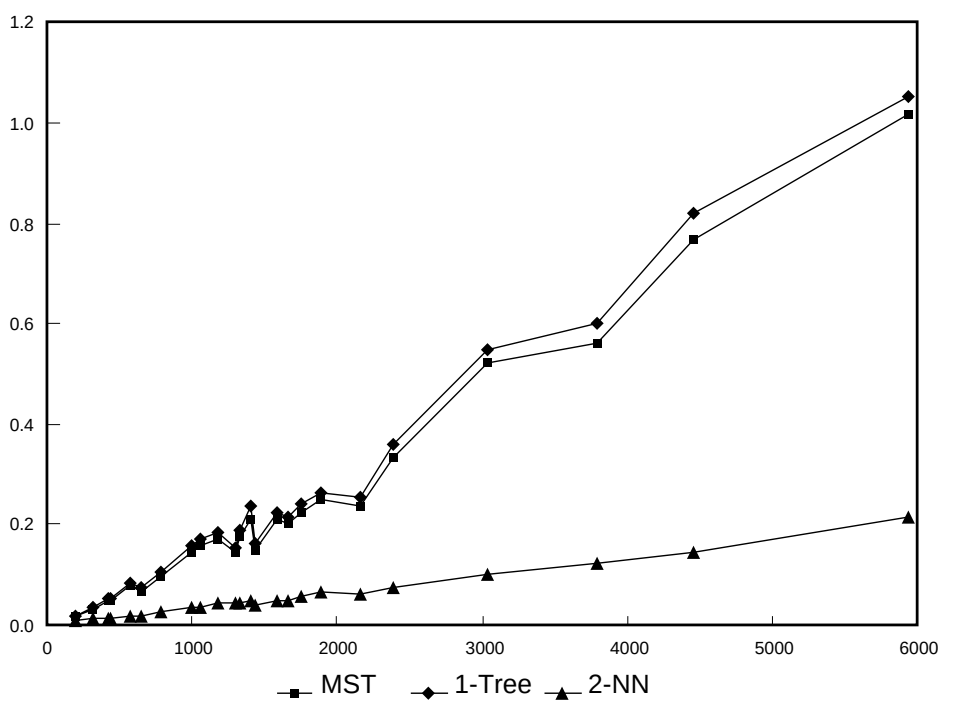

Figure 10.6 CPU times for tree lower bounds

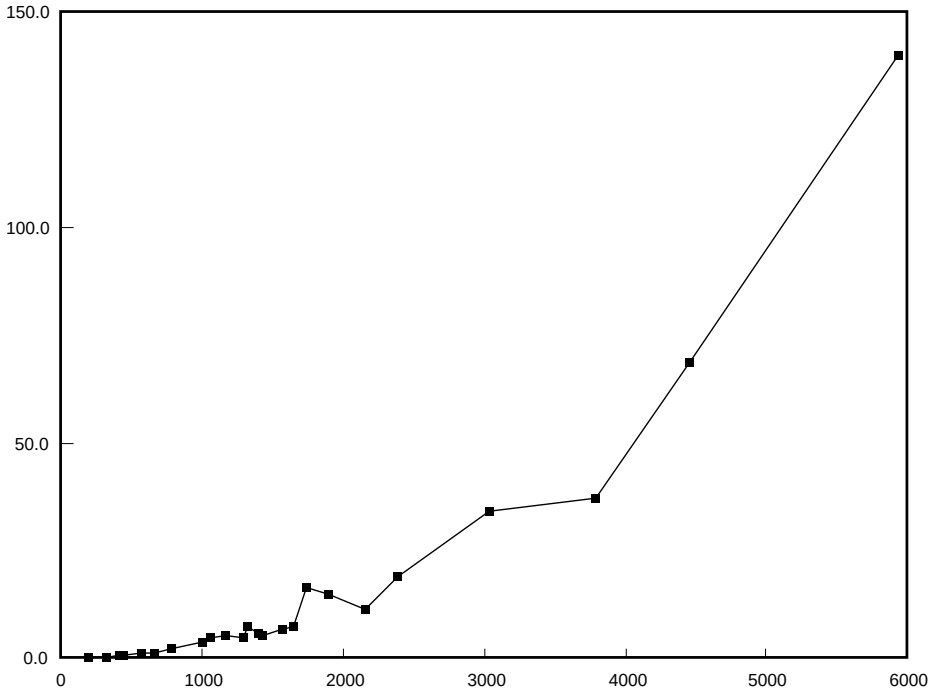

Figure 10.7 CPU times for assignment lower bounds

Figure 10.7 shows the running times for our implementation of the Hungarian method for solving assignment problems. Running time, even for the sparse graphs, are considerable. Therefore, the assignment bound is, not only because of its weakness, of no interest for practical computations.

10.3 Lagrangean Relaxation

The 1-tree and the 2-neighbor bounds discussed in the previous sections are useful for practical purposes, but are quite weak as the experiments have exhibited. In this section we discuss how these bounds can be improved by fairly simple means requiring only little additional implementational effort.

Before putting this bound improvement into a proper theoretical context, we will describe an intuitive approach. In a tour exactly two edges are incident to each node. If we associate with each node i some weight π_i and use modified edge weights $c'_{ij} = c_{ij} + \pi_i + \pi_j$ then the length of every tour is increased by $2 \cdot \sum_{i=1}^{n} \pi_i$. Hence the relative order of the tours with respect to their length remains unchanged. If we compute lower bounds using the new weights and subtract $2 \cdot \sum_{i=1}^{n} \pi_i$ we obtain lower bounds for the original problem. Therefore we can use the node weights to make nodes more or less attractive to try, for example, to approach the satisfaction of the degree constraints in 1-tree computations. Nodes having degree 1 in the current 1-tree should become more attractive whereas edges to nodes with high degree should receive larger weight. Simple examples show already that bounds can indeed be improved this way.

This intuitive idea is exactly reflected in Lagrangean relaxation approaches to the TSP. We will first describe the general method.

10.3.1 The General Approach

One of the most common approaches for obtaining bounds on the optimal objective function value of an integer linear program is the method of **Lagrangean relaxation**. Let the following integer linear programming problem be given.

$$\text{(P)} \qquad \begin{aligned} z^* := \quad & \min\, c^T x \\ & Ax \;= b \\ & Bx \;= d \\ & x \;\geq 0 \\ & x \text{ integer.} \end{aligned}$$

For any y the following integer linear programming problem provides a lower bound for the minimal value z^* of (P).

$$\text{(D}_y\text{)} \qquad \begin{aligned} L(y) := \quad & \min\, c^T x + (d - Bx)^T y \\ & Ax \;= b \\ & x \;\geq 0 \\ & x \text{ integer.} \end{aligned}$$

The lower bound property can easily be seen since every feasible solution for (P) is also feasible for (D$_y$) with the same objective function value. Note that, since y is not restricted in sign, we could as well use the objective function $c^T x + (Bx - d)^T y$. The vector y is called vector of **Lagrange multipliers**. The best such lower bound is then given by solving the so-called **Lagrangean dual problem**

$$\text{(LD)} \qquad u^* := \quad \max_y L(y)$$

i.e., by finding the maximum of the function L.
The function L is piecewise linear and concave (and hence nondifferentiable). A suitable method for maximizing L is **subgradient maximization**. A vector d is called **subgradient** of L at x, if $d^T(y - x) \geq L(y) - L(x)$ for all y.
Suppose x^* is the minimizer of D$_{y^*}$ for some vector of Lagrange multipliers y^*. Then $u^* = d - Bx^*$ is a subgradient of L at y^* as is seen from

$$\begin{aligned} L(y^* + h) - L(y^*) &\leq c^T x^* + (d - Bx^*)^T (y^* + h) - c^T x^* - (d - Bx^*)^T y^* \\ &= (d - Bx^*)^T h \\ &= h^T u^*, \quad \text{for all } h. \end{aligned}$$

Hence solving problem D$_y$ for some y, at the same time provides a subgradient of L at y.
The **subgradient method** is an iterative method which at a given point y^k computes the next iterate y^{k+1} by

$$y^{k+1} = y^k + \lambda_k d^k,$$

where d^k is a subgradient of L at y^k and λ_k is a suitable step length.
If L is bounded from above and if the step lengths satisfy both $\lim_{k\to\infty} \lambda_k = 0$ and $\sum_{k=0}^{\infty} \lambda_k = \infty$, then the method converges to the maximum of L (POLYAK (1978)).
It turned out in practice that the step length formula above leads to very slow convergence. So the requirement $\sum_{k=0}^{\infty} \lambda_k = \infty$ is usually dropped and there are several formulas for computing the step length leading to satisfactory convergence in practical applications. A widely used formula is e.g.,

$$\lambda_k = \alpha \cdot \frac{U - L(y^k)}{||d^k||},$$

where U is an upper bound on L, and $0 < \alpha < 2$ is some constant which is periodically decreased. The method is stopped as soon as there is no more significant increase in the bound.
Clearly, the quality of the bound provided by the Lagrangean dual depends on the choice of the constraint set $Bx = d$ to be relaxed. This also influences the complexity of the Lagrangean subproblems to be solved for each multiplier set y.
Consider the following series of problems.

$$\text{(P)} \qquad z^* := \min_x\{c^T x \mid Ax = b, Bx = d, x \geq 0, x \text{ integer }\}$$

$$(\text{P}_{\text{LP}}) \quad z^*_{LP} := \min_x\{c^T x \mid Ax = b, Bx = d, x \geq 0\}$$

$$z^*_D := \max_{u,v}\{d^T u + b^T v \mid u^T B + v^T A \leq c^T\}$$

$$u^*_D := \max_u\{d^T u + \max_v\{b^T v \mid v^T A \leq c^T - u^T B\}\}$$

$$u^*_{LP} := \max_u\{d^T u + \min_x\{(c^T - u^T B)x \mid Ax = b, x \geq 0\}\}$$

$$\text{(LD)} \qquad u^* := \max_u\{d^T u + \min_x\{(c^T - u^T B)x \mid Ax = b, x \geq 0, x \text{ integer }\}\}$$

In general we have $z^* \geq z^*_{LP} = z^*_D = u^*_D = u^*_{LP} \leq u^*$.
In the special case where $\{x \mid Ax = b, x \geq 0\}$ is an integer polyhedron (i.e., has only integer vertices), we have $u^* = u^*_{LP} = z^*_{LP}$. In this case, the value of the Lagrangean dual is equal to the value of the linear programming relaxation (P_{LP}) of the integer linear program (P). Moreover, as noted in SCHRIJVER (1986), if we can solve $\min\{(c^T - u^T B)x \mid Ax = b, x \geq 0\}$ in polynomial time for each u, then we can also solve (LD) in polynomial time and we can directly apply LP techniques on (P_{LP}) to compute u^*. But, even though (LD) might be solvable in polynomial time, this fact might not be exploitable in practice, especially when larger problems have to be solved. Therefore, for each choice of Lagrangean relaxation one has to think about the best way for computing or approximating u^*.

For convenience we have used equation systems for both the relaxed and the maintained constraints. Treating inequality systems does not cause any problems. If inequalities ("$\geq$") are relaxed then their corresponding Lagrange multipliers must be nonnegative. We will now show that 1-tree and 2-neighbor relaxations are particular applications of the general method for the TSP.

10.3.2 Lagrangean Relaxation with 1-Trees

In section 2.3 we have given an integer linear programming formulation of the TSP. For the purposes of this chapter we write this formulation in a different, but equivalent form.

$$
\begin{aligned}
c^* := \quad \min \sum_{ij \in E_n} c_{ij} x_{ij} & \\
\sum_{j=1}^{n} x_{ij} &= 2, && \text{for } i \in V_n \setminus \{1\}, \\
\sum_{j=1}^{n} x_{1j} &= 2, && \\
\sum_{ij \in E_n} x_{ij} &= n, && \\
x(C) &\leq |C| - 1, && \text{for all cycles } C \text{ in } \{2, 3, \ldots, n\}, \\
x_{ij} &\in \{0, 1\}, && \text{for all } ij \in E_n.
\end{aligned}
$$

If we now relax the first system of equations and associate multipliers π_i with the nodes, we obtain the following relaxation. For convenience we also define π_1 and set it to 0.

$$
\begin{aligned}
L(\pi) := \quad \min \; -2\sum_{i=1}^{n} \pi_i + \sum_{ij \in E_n} (c_{ij} + \pi_i + \pi_j) x_{ij} & \\
\sum_{j=1}^{n} x_{1j} &= 2, && \\
\sum_{ij \in E_n} x_{ij} &= n, && \\
x(C) &\leq |C| - 1, && \text{for all cycles } C \text{ in } \{2, 3, \ldots, n\}, \\
x_{ij} &\in \{0, 1\}, && \text{for all } ij \in E_n.
\end{aligned}
$$

It is known that the condition $x_{ij} \in \{0, 1\}$, for all $ij \in E_n$, is not necessary in this formulation because the system of linear equations and inequalities describes an integral polyhedron whose vertices are exactly the incidence vectors of 1-trees. Therefore, due to the observations above, the value of the Lagrangean dual based on 1-trees is equivalent to the lower bound provided by the subtour elimination relaxation defined in 10.1.2. Since feasible solutions of the integer programm are exactly 1-trees with special node 1, the relaxation can be rewritten as follows.

$$L(\pi) := \quad \min \sum_{ij \in E_n} c_{ij}x_{ij} + \sum_{i=1}^{n} \pi_i \Big(\sum_{j=1}^{n} x_{ij} - 2 \Big)$$

$$x \text{ is incidence vector of a 1-tree .}$$

Determining $L(\pi)$ for a given π amounts to computing a 1-tree with respect to the modified edge weights $c_{ij}+\pi_i+\pi_j$ and subtracting $2\sum_{i=1}^{n} \pi_i$. According to the preceding section, the minimum 1-tree readily supplies a subgradient as follows. Let δ_i be the degree of node i in the minimum 1-tree. Then the vector $(\delta_1 - 2, \delta_2 - 2, \ldots, \delta_n - 2)$ is a subgradient of L at π.
HELD & KARP (1970,1971) describe the first algorithm for finding the maximum of L, and we therefore call the best 1-tree bound also **Held-Karp bound**. There are several variations concerning the choice of initial step sizes and update of step sizes (HELBIG-HANSEN & KRARUP (1974), SMITH & THOMPSON (1977), VOLGENANT & JONKER (1982), and BALAS & TOTH (1985)). Based on these references and on own experiments we used the following implementation.

procedure 1tree_bound

(1) Let τ be the initial step length, λ a decrement factor for the step length, and m the number of iterations.

(2) Set $t^1 = \tau$, $\pi_i^1 = 0$ for every node i, and $k = 1$.

(3) As long as $k \leq m$ perform the following steps.

(3.1) Compute a minimum spanning tree with respect to the edge weights $c_{ij} + \pi_i + \pi_j$.

(3.2) Compute the best 1-tree obtainable from this spanning tree (section 10.2).

(3.3) Define the vector d^k by $d_i^k = \delta_i - 2$, where δ_i is the degree of node i in the 1-tree computed in Step (3.2).

(3.4) For every node i set

$$\pi_i^{k+1} = \pi_i^k + t^k(0.7d_i^k + 0.3d_i^{k-1}).$$

(3.5) Set $t^{k+1} = \lambda t^k$ and increment k by 1.

(4) Return the best bound computed.

end of 1tree_bound

Differences to straightforward realizations are, that the direction of the subgradient step is a convex combination of the current and the preceding subgradient, that the direction vector is not normalized, and that the special node for the 1-tree computations is not fixed. In theory, the same optimal value of the Lagrange dual is attained whatever node is fixed. Actually, it is even incorrect to have varying nodes because the underlying optimization problem changes. But practical experiments have shown that better bounds

are obtained this way and that it pays off to spend the additional running time for computing various 1-trees.
Some authors propose to update the multipliers according to the formula

$$\pi_i^{k+1} = \pi_i^k + t^k(U - L(\pi^k))\frac{d_i^k}{||d^k||}$$

where U is an estimate for the optimal solution and $L(\pi^k)$ is the currently computed lower bound. We found that convergence is faster with this formula, but bounds are inferior.
A general guideline for the performance is the following. If λ is close to 1 (say 0.98–0.995) convergence is slow, but usually better bounds are reached. For values of λ between 0.95 and 0.97 faster convergence is achieved yielding reasonable bounds. Smaller values of λ lead to considerably inferior bounds. Instead of fixing the number of iterations one can stop, if no significant progress is observed any more.
Since no line search is performed it is not guaranteed that each iteration step improves the bound. In fact, the behaviour shown in Figure 10.8 can be observed. This figure displays the development of the bounds during application of the above subgradient method to problem `pcb442` for 150 iterations.

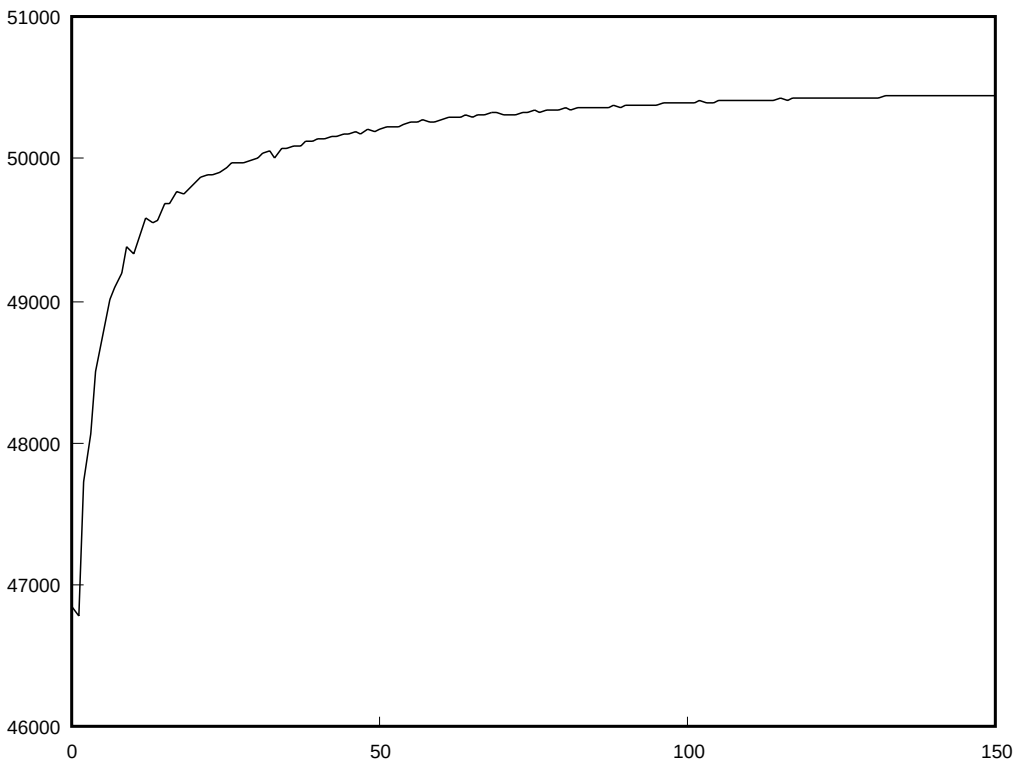

Figure 10.8 A run of the subgradient algorithm

The best bound obtained is 50459. In general, the evolution of the lower bounds does not have to be as smooth as in this example. Depending on the problem instance the

use of non-fixed special nodes can have the consequence that the bounds alternate even at the end of the procedure. But, our emphasis does not lay on nice convergence but on good bounds. Hence this is tolerable.
Concerning running time we are in a bad situation. Since edge weights are arbitrary we can compute the best tree in Step (3.1) only in time $O(n^2)$ and the best 1-tree in Step (3.2) in time $O(kn)$ where k is the number of leaves of the spanning tree.
To speed up the procedure we compute trees in sparse subgraphs. This has the consequence that the computed bound may not be valid for the true problem. Only if the subgraph contains an optimal tour for the original problem, the bound is valid. Of course, this cannot be verified. We therefore proceeded as follows for our sample problems.

procedure fast_1treebound

(1) Construct the subgraph consisting of the 10 nearest neighbor edges and the edges of the Delaunay graph.

(2) Perform the subgradient algorithm only in this graph to compute an approximate lower bound.

(3) Compute a minimum 1-tree in the complete graph using the multipliers of the final iteration in (2).

end of fast_1treebound

Problem	Subgraph 150 It.	Subgraph 300 It.	Final iteration
d198	6.92	5.31	5.65
lin318	2.34	0.61	0.61
fl417	6.37	3.31	3.50
pcb442	2.55	0.63	0.63
u574	2.62	0.55	0.57
p654	10.00	4.19	4.23
rat783	2.13	0.40	0.41
pr1002	3.40	0.99	0.99
u1060	3.10	0.87	0.87
pcb1173	2.19	0.97	0.99
d1291	2.37	1.48	1.70
rl1323	1.98	1.59	1.72
fl1400	10.15	2.42	6.38
u1432	3.31	0.46	0.47
fl1577	6.19	5.34	5.40
d1655	2.34	1.24	1.24
vm1748	2.72	1.37	1.37
rl1889	2.29	1.61	1.74
u2152	2.08	0.55	0.60
pr2392	2.52	1.23	1.23
pcb3038	2.44	0.84	0.84
fl3795	7.13	4.46	4.61
fnl4461	2.49	0.58	0.58
rl5934	1.63	1.20	1.23
Average	3.80	1.76	1.98

Table 10.9 Results of Lagrangean relaxation based on 1-trees

Running time is now $O(n \log n) + O(kn)$ for each iteration in Step (2) where k is the number of leaves of the spanning tree and $O(n^2)$ for the final step. Due to this final optimization a valid lower bound is determined.

Table 10.9 documents a sample run with this fast bounding procedure where we have set $\lambda = 0.98$ and $m = 300$. As initial step length we choose $10 \cdot (U - T)/n$ where U is the tour length shown in Table 6.17 for the Christofides starting tour and T is the 1-tree bound listed in Table 10.5.

We give the bounds obtained from the subgraph optimization after 150 and after 300 iterations and the bound obtained in the final 1-tree computation. The table verifies that our approach is indeed reasonable. The final iteration changes the bound only marginally. In practical applications we can safely omit the final step and assume that the bound determined in the first phase is correct.

Respective CPU times are given in Figure 10.10. They do not increase smoothly because the running time of Kruskal's algorithm depends on the distribution of edge lengths for the respective subgraphs. This distribution influences the number of edges that are checked for entering the spanning tree. A uniform distribution usually leads to earlier termination.

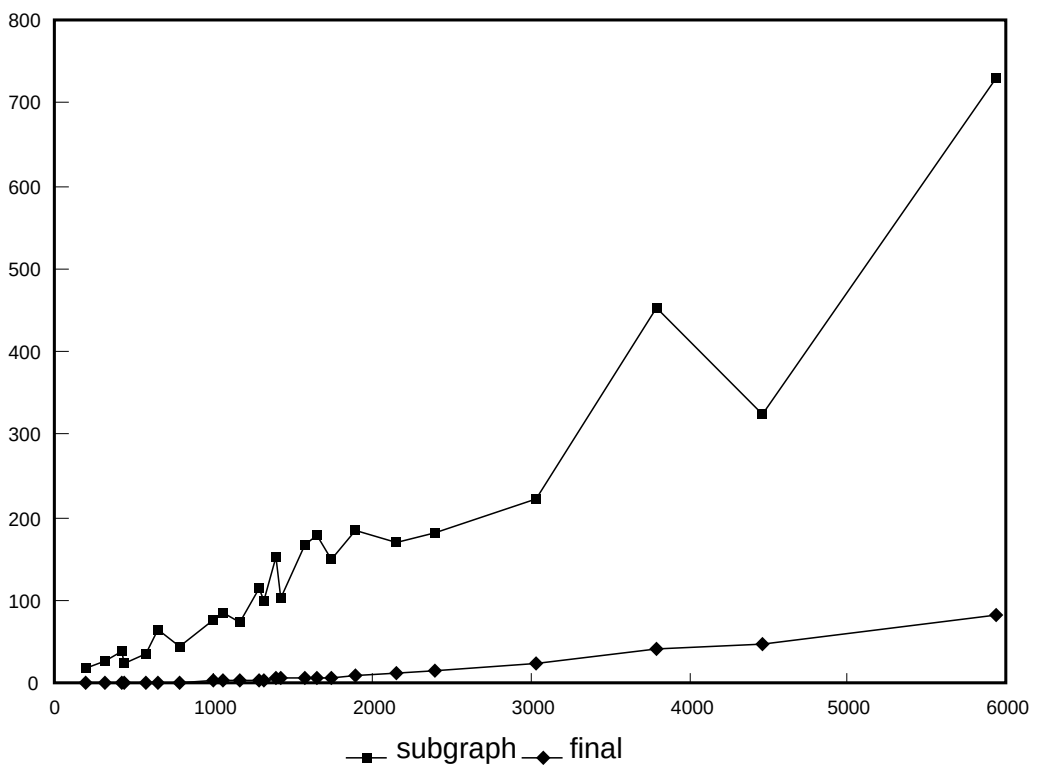

Figure 10.10 CPU times for Lagrangean relaxation (1-tree)

Methods for finding the optimum of the Lagrangean dual based on 1-trees are not limited to subgradient approaches. One further iterative method is, e.g., dual ascent (Malik & Fisher (1990)). As note above, a completely different way is to directly solve the linear program corresponding to the subtour elimination relaxation.

10.3.3 Lagrangean Relaxation with 2-Neighbor Configurations

We will now show that the 2-neighbor configurations are related to the fractional 2-matching relaxation. It is known (BALINSKI (1970)) that the polytope defining the fractional 2-matching problem has only vertices whose components have values in $\{0, \frac{1}{2}, 1\}$. To treat 2-neighbor configurations we switch to (directed) arcs. We introduce variables y_{ij} with the interpretation that $y_{ij} = 1$ if the arc from i to j is selected, and $y_{ij} = 0$ otherwise. We set also $c_{ij} = c_{ji}$ and for notational convenience, we use variables y_{ii} in the formulae (which then have to be ignored).
Consider the following integer programming problem

$$
\begin{aligned}
\min \frac{1}{2} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}y_{ij} & \\
\sum_{j=1}^{n} y_{ij} &= 2, \qquad \text{for all } i \in V_n, \\
\sum_{j=1}^{n} y_{ji} &= 2, \qquad \text{for all } i \in V_n, \\
y_{ij} &\in \{0,1\}, \quad \text{for all } i,j \in V_n.
\end{aligned}
\tag{F}
$$

If y^* is a feasible solution of this problem then x^* defined via $x^*_{ij} = \frac{1}{2}(y^*_{ij} + y^*_{ji})$ is a feasible solution of the fractional 2-matching problem. If x^* with $x^*_{ij} \in \{0, \frac{1}{2}, 1\}$ is feasible for the latter problem then $y^*_{ij} = y^*_{ji} = x^*_{ij}$ is feasible for the new problem.
If we formulate a Lagrangean relaxation approach for problem (F) by relaxing the second set of constraints we obtain the following problem

$$
\begin{aligned}
L(\pi) := \quad \min \frac{1}{2} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}y_{ij} + \sum_{i=1}^{n} \pi_i \big(\sum_{j=1}^{n} y_{ji} - 2\big) & \\
\sum_{j=1}^{n} y_{ij} &= 2, \qquad \text{for all } i \in V_n, \\
y_{ij} &\in \{0,1\}, \quad \text{for all } i,j \in V_n.
\end{aligned}
$$

Evaluating $L(\pi)$ for a given π amounts to computing the minimum 2-neighbor configuration for modified edge weights. The new edge weights c'_{ij} are obtained as $c'_{ij} = \frac{1}{2}c_{ij} + \pi_j$ (note that in general $c'_{ij} \neq c'_{ji}$). Solving the Lagrangean dual, i.e., maximizing L gives the fractional 2-matching bound. Concerning the implementation of a subgradient method, the same remarks as for the 1-tree relaxation apply.
Running time is $O(n^2)$ for each iteration of the subgradient method. To speed up computations,we optimize in subgraphs only. If the subgraph has m edges each iteration takes time $O(m)$ and the final iteration takes time $O(n^2)$ to yield a valid lower bound. We ran experiments using the 10 nearest neighbor subgraph augmented by the edges of the Delaunay graph. In this case each iteration runs in time $O(n)$. Table 10.11 shows the results. Again, the final step alters the bounds only slightly and the approach is verified to be reasonable.

Problem	Subgraph 150 It.	Subgraph 300 It.	Final iteration
lin318	7.50	7.43	7.43
fl417	22.68	22.50	24.37
pcb442	1.63	1.34	1.34
u574	7.28	7.18	7.18
p654	18.39	17.53	17.71
rat783	2.83	2.70	2.70
pr1002	7.09	7.01	7.02
u1060	6.74	6.51	6.51
pcb1173	2.35	2.27	2.27
d1291	7.60	7.54	7.54
rl1323	9.37	9.27	9.28
fl1400	16.72	15.52	15.54
u1432	1.45	0.88	0.88
fl1577	19.48	19.28	19.28
d1655	5.37	5.24	5.24
vm1748	4.58	4.46	4.49
rl1889	9.23	9.11	9.12
u2152	4.45	4.22	4.22
pr2392	4.99	4.87	4.87
pcb3038	1.77	1.67	1.67
fl3795	16.09	15.49	15.49
fnl4461	1.90	1.82	1.82
rl5934	5.92	5.86	5.86
Average	8.79	8.54	8.63

Table 10.11 Results of Lagrangean relaxation based on 2-neighbors

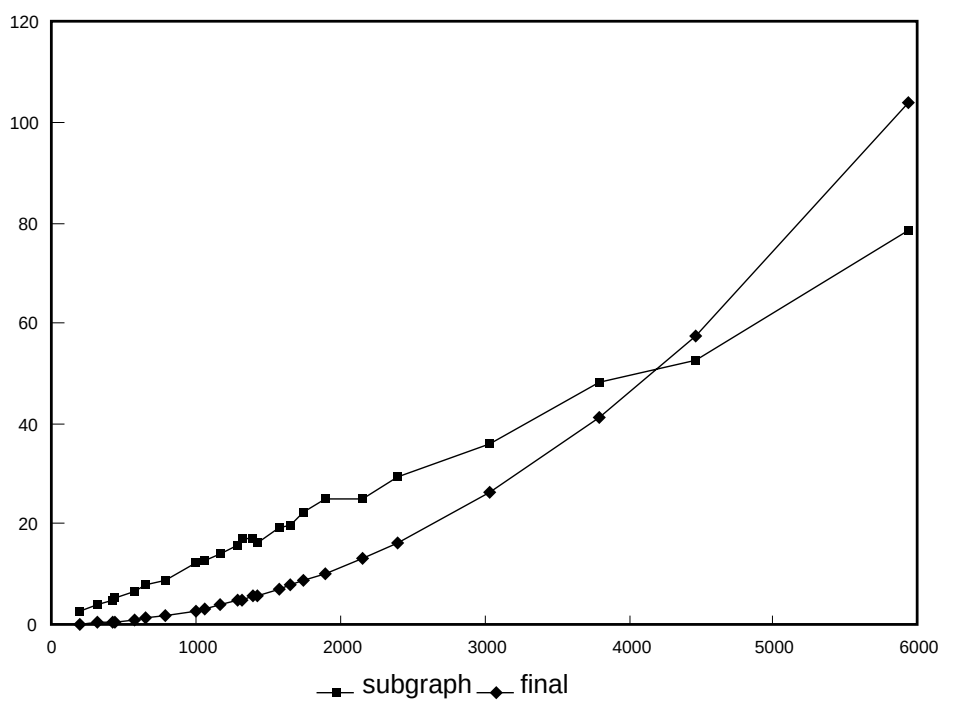

Figure 10.12 CPU times for Lagrangean relaxation (2-neighbor)

CPU time statistics show that time per iteration is considerably less for the 2-neighbor relaxation than for the 1-tree relaxation. Therefore, if one observes the development of the bounds over time, for some problems during a certain first time period the 2-neighbor bound will we above the 1-tree bound. In the end, according to the theoretical results, the 1-tree bound will be better.
Lagrangean relaxations based on perfect 2-matchings are considered in SMITH, MEYER & THOMPSON (1990).

10.3.4 Multiplier Heuristics

Every assignment of node multipliers can be used to determine a lower bound using 1-trees or 2-neighbor configurations. The simple lower bounds implicitly use zero multipliers, the subgradient method adapts the multipliers starting with zero multipliers. In this section we briefly address the question of using a heuristic for guessing reasonable multipliers.
To get an impression on how the π_i values look like we show in Figure 10.13 near optimal multipliers for the 1-tree relaxation for problem `pcb442` giving the lower bound 50490. Figure 10.14 displays multipliers for the 2-neighbor relaxation giving a lower bound of 50099.

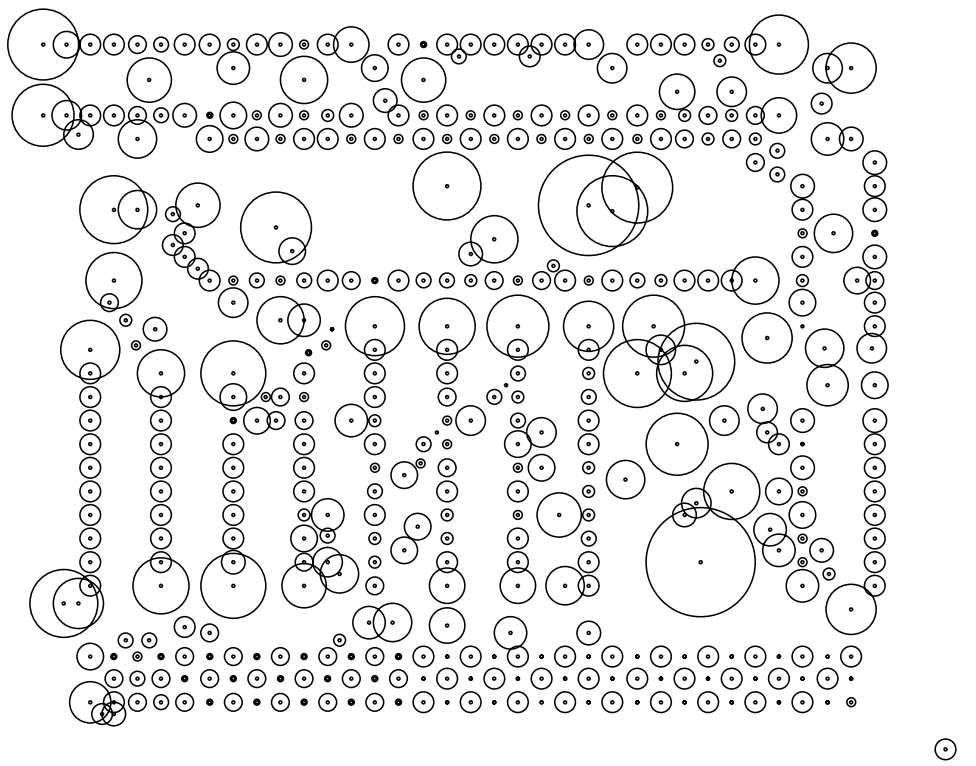

Figure 10.13 Near optimal 1-tree multipliers for `pcb442`

Note that one can add the same constant to all multipliers without affecting the lower bounds obtained by the relaxation. We have therefore subtracted the maximum π_i

from all multipliers. Now all values are nonpositive and the largest one is zero. Radii of circles in the figures are the complemented π_i values.
The only hint for estimating reasonable multipliers is that apparently isolated nodes have large negative multipliers whereas the other nodes have multipliers related to their nearest neighbor distance. This corresponds to the intuitive explanation that isolated nodes have to be made more attractive since otherwise they would become leaves of the 1-tree.

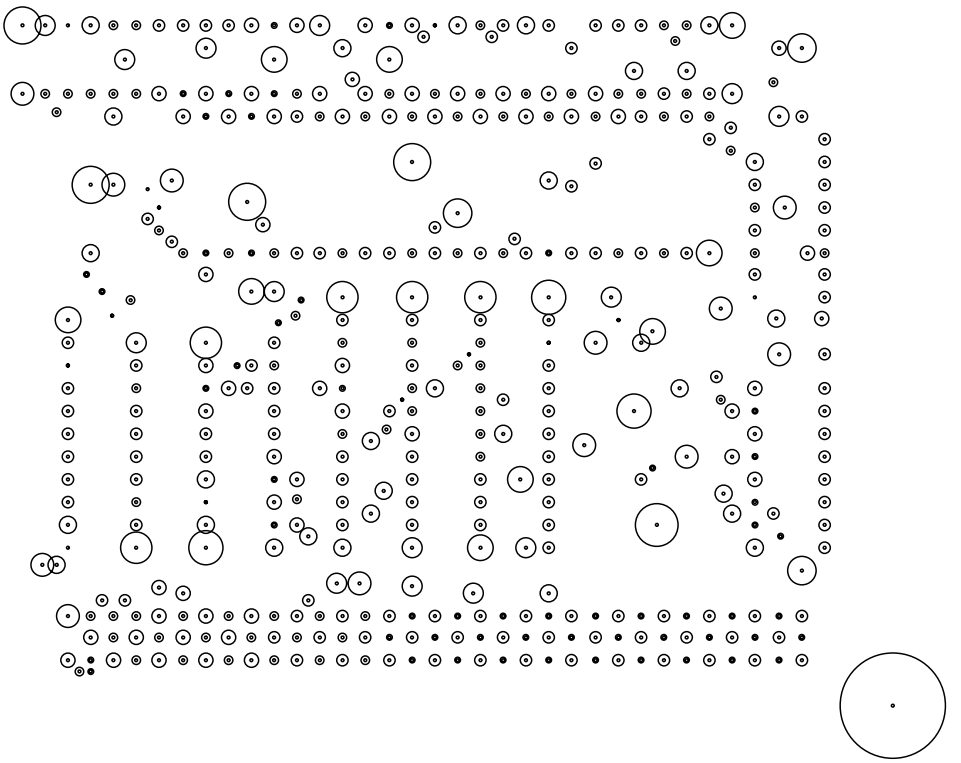

Figure 10.14 Near optimal 2-neighbor multipliers for `pcb442`

We have implemented three variants of multiplier heuristics which also run fast. Let $d_1(i)$ be the distance to the nearest neighbor of i and $d_2(i)$ be the distance to the second nearest neighbor. The variants are

(1) Set $\pi_i = -0.5d_1(i)$.

(2) Set $\pi_i = -0.5d_2(i)$.

(3) Set $\pi_i = -0.25(d_1(i) + d_2(i))$.

We have seen in preceding chapters that nearest neighbor distances for Euclidean problems can be computed very efficiently. For the general problem we need time $O(n^2)$ to compute these multipliers. In any case, eve if multipliers can be computed fast, we have to spend time $O(n^2)$ for proving the validity of the bounds.
Table 10.15 displays the results obtained with Variant 2 for the 1-tree and the 2-neighbor relaxation which are significantly better than the simple bounds of Table 10.5. The other two variants performed worse, so their results are not documented.

Problem	1-tree	2-nn
d198	15.96	26.76
lin318	6.83	12.32
fl417	12.78	26.52
pcb442	4.53	2.51
u574	9.11	10.02
p654	10.70	19.78
rat783	5.28	5.30
pr1002	9.13	9.36
u1060	8.28	9.24
pcb1173	5.56	4.19
d1291	4.53	9.17
rl1323	9.69	12.79
fl1400	9.23	19.36
u1432	2.96	2.26
fl1577	11.69	20.73
d1655	5.01	6.91
vm1748	7.48	7.66
rl1889	10.33	12.36
u2152	3.18	5.57
pr2392	6.92	7.49
pcb3038	4.34	3.34
fl3795	8.91	16.41
fnl4461	4.57	3.70
rl5934	6.31	8.56
Average	7.64	10.93

Table 10.15 Results of multiplier heuristics

Further aspects of Lagrangean relaxation for the TSP are discussed in SHMOYS & WILLIAMSON (1990) and SMITH, MEYER & THOMPSON (1990).

10.4 Comparison of Lower Bounds

We have seen that the optimal objective function value c_T of the Lagrangean dual based on 1-trees is the subtour elimination bound. The Lagrangean dual based on 2-neighbor configurations gives the fractional 2-matching bound c_N. Since the latter is itself a relaxation of the subtour relaxation we get that $c_N \leq c_T$. Normally, we have $c_N < c_T$. For practical problems the difference between the two bounds will be considerable.
To examine how well our subgradient algorithm approximates the optima c_N and c_T we have also computed the exact values using LP techniques. We used an branch and cut code for solving TSPs optimally (JÜNGER, REINELT & THIENEL (1993)) to compute these bounds for all of our sample problems. Table 10.16 displays the results and shows that in some cases (e.g., d198, p654, or rl1323) the 1-tree bound computed in Table 10.9 misses the optimal 1-tree bound by some percent. Having a look at such problems, one realizes that they are built of clusters of points. For such instances, our subgradient method has problems in approaching the best bound. We can only overcome this by enlarging the decrement factor and hence coming closer to the theoretically required formula for obtaining convergence to the optimum of the Lagrangean dual. For example, if we use $\lambda = 0.995$, then for problem d198 we obtain the lower bound 14769 after 800

iterations. If the points are more or less uniformly distributed we have no difficulties in finding rather close approximations to the best bound.

Problem	Fractional 2-Matching	Subtour elimination
d198	11793	15712
lin318	38964	41889
fl417	8961	11790
pcb442	50104	50500
u574	34256	36714
p654	28584	34596
rat783	8568	8773
pr1002	240878	256766
u1060	209529	222651
pcb1173	55600	56351
d1291	46971	50209
rl1323	245156	265815
fl1400	16783	19783
u1432	151676	152535
fl1577	17871	21886
d1655	58876	61544
vm1748	321552	332061
rl1889	287698	311705
u2152	61464	63859
pr2392	359620	373490
pcb3038	135391	136588
fl3795	24289	28478
fnl4461	179252	181570
rl5934	521629	548471

Table 10.16 Exact values of relaxations

Finally, in Table 10.17, we give the average qualities of all relaxations discussed in this chapter taking only those 19 sample instances into account where true optimal solutions are known.

Name of heuristic	Average deviation from optimum
Subtour Elimination	0.78
Lagrange 1-Tree bound (final step)	1.54
Multiplier heuristic (1-Tree)	7.58
Fractional 2-Matching	7.70
Lagrange 2-NN bound (final step)	7.72
Geometric bound	9.70
Simple 1-tree	9.93
Multiplier heuristic (2-NN)	10.09
Minimum spanning tree	11.15
Simple 2-neighbors	15.69
Assignment	18.90

Table 10.17 Comparison with optimal solutions

The table shows that the fractional 2-matching bound is about 7% below the subtour elimination bound. Our fast schemes to approximate the subtour bound and the

fractional 2-matching bound perform fairly well on the average. Whereas the fractional 2-matching bound is almost met by our heuristic, the subtour bound is missed by about 1%. As we pointed out, we can improve the approximation by tuning the parameters of the subgradient algorithm.

Multiplier heuristics improve the simple bounds considerably. If Delaunay graphs are available, then at least the simple bound computations should be performed in any case.

The subgradient method used in this chapter is not the only way for attacking nondifferentiable optimization problems. A more elaborate approach is the so-called **bundle method** (KIWIEL (1989)). It is also based on subgradients, but in every iteration the new direction is computed as a convex combination of several (10–20) previous subgradients. Moreover, line searches are performed. In this sense, our approach is a simple version of the bundle method keeping only a "bundle" of two subgradients (which are combined in a fixed way) and not performing line searches.

SCHRAMM (1989) considers an extension of this principle which combines the bundle approach with trust-region methods. Whereas, in general, this algorithm outperforms pure subgradient methods this is not the case for the 1-tree relaxation. Here performance is similar.

Several further relaxations are available for the TSP. Among them are **n-path relaxation** or so-called **additive bounding** procedures. For information on further approaches see HOUCK, PICARD, QUEYRANNE & VEMUGANTI (1980), BALAS & TOTH (1985), MACULAN & SALLES (1989). CARPANETO, FISCHETTI & TOTH (1989), and FISCHETTI & TOTH (1992).