In general, we will not end up with $L(w) = w(\text{TSP})$: it is quite possible that no choice of $\mathbf{p}$ yields a minimal s-tree which is already a tour; an example for this situation can be found in [HeKa70]. But the lower bound for $w(\text{TSP})$ given by (L) is particularly strong: the values of $L(w)$ are on average more than 99% of $w(\text{TSP})$ according to [VoJo82]. An interesting theoretical examination of the Held-Karp technique can be found in [ShWi90].

Of course, solving (L) is a considerably more involved problem than the original s-tree relaxation. There are various approaches to this problem; the vectors $\mathbf{p}$ are called *subgradients* in this context. These subgradients can be used for solving (L) recursively; this yields a method which is guaranteed to converge to $L(w)$ (for an appropriate choice of the step widths c). Unfortunately, one cannot predict how many steps will be required, so that the process is often terminated in practice as soon as the improvement between successive values becomes rather small.

The problem (L) is a special case of a much more general method which is used quite often for integer linear programming problems: *Lagrange relaxation*; we refer to [Sha79] and [Fis81]. The approach via *subgradient optimization* is only one of several ways to solve Lagrange relaxations; it is described in detail (together with other methods) in [Sho85]; see also [HeWC74].

Appropriate relaxations are very important for finding the optimal solution of a TSP, because they form an essential part of *branch-and-bound* techniques; we will present an example for such a method in Section 15.8. We refer the reader to [VoJo82] and to [LaLRS85, Chapter 10] for more detailed information. Further methods for determining lower bounds can be found, for example, in [CaFT89].

15.4 Approximation algorithms

The preceding two sections treated the problem of finding lower bounds on the length of an optimal tour, so it is now natural to ask for upper bounds. It would be nice to have an algorithm (of small complexity, if possible) for constructing a tour which always gives a provably good approximation to the optimal solution. We need a definition to make this idea more precise, which generalizes the approach we took when we studied the greedy algorithm as an approximation method in Section 5.4.

Let $\mathbf{P}$ be an optimization problem, and let $\mathbf{A}$ be an algorithm which calculates a feasible – though not necessarily optimal – solution for any given instance I of $\mathbf{P}$. We denote the weights of an optimal solution and of the solution constructed by $\mathbf{A}$ by $w(I)$ and $w_{\mathbf{A}}(I)$, respectively. If the inequality

$$|w_{\mathbf{A}}(I) - w(I)| \leq \varepsilon w(I) \tag{15.9}$$

holds for each instance I, we call $\mathbf{A}$ an *ε-approximative algorithm* for $\mathbf{P}$. For example, a 1-approximative algorithm for the TSP would always yields a tour which is at most twice as long as an optimal tour.

Given an NP-complete problem, there is little hope to find a polynomial algorithm which solves $\mathbf{P}$ correctly. Thus it seems promising to look instead for a polynomial ε-approximative algorithm, with ε as small as possible. Unfortunately, this approach is often just as difficult as solving the original problem. In particular, this holds for the TSP, as the following result of Sahni and Gonzales [SaGo76] shows.

Theorem 15.4.1. *If there exists an ε-approximative polynomial algorithm for the TSP, then $P = NP$.*

HC is NP-complete

Proof. Let $\mathbf{A}$ be an ε-approximative polynomial algorithm for the TSP. We will use $\mathbf{A}$ to construct a polynomial algorithm for determining a Hamiltonian cycle; then the assertion follows from Theorem 2.7.4. The construction resembles the one given in the proof of Theorem 2.7.5. Let $G = (V, E)$ be a connected graph, and consider the complete graph K_V on V with weights

$$w_{ij} = \begin{cases} 1 & \text{for } ij \in E \\ 2 + \varepsilon|V| & \text{otherwise.} \end{cases}$$

If the given algorithm $\mathbf{A}$ should determine a tour of weight $n = |V|$ for this instance of the TSP, then G is obviously Hamiltonian.

Conversely, suppose that G contains a Hamiltonian cycle. Then the corresponding tour has weight n and is trivially optimal. As $\mathbf{A}$ is ε-approximative by hypothesis, it will compute a tour π of weight $w(\pi) \leq (1 + \varepsilon)n$. Suppose that π contains an edge $e \notin E$. Then

$$w(\pi) \geq (n - 1) + (2 + \varepsilon n) = (1 + \varepsilon)n + 1,$$

a contradiction. Hence the tour π determined by $\mathbf{A}$ actually induces a Hamiltonian cycle in G, so that it has in fact weight n.

We have proved that G is Hamiltonian if and only if $\mathbf{A}$ constructs a tour of weight n for our auxiliary TSP, so that $\mathbf{A}$ would indeed yield a polynomial algorithm for HC. □

Clearly, a result analogous to Theorem 15.4.1 holds for the ATSP. Interestingly, the situation is much more favorable for the metric TSP. We need a definition and a lemma. Let K_n be the complete graph on $V = \{1, \ldots, n\}$. Then any connected Eulerian multigraph on V is called a *spanning Eulerian multigraph* for K_n.

Lemma 15.4.2. *Let W be the weight matrix of a ΔTSP on K_n, and let $G = (V, E)$ be a spanning Eulerian multigraph for K_n. Then one can construct with complexity $O(|E|)$ a tour π satisfying $w(\pi) \leq w(E)$.*

Proof. By Example 2.5.3, it is possible to determine with complexity $O(|E|)$ an Euler tour C for G. Write the sequence of vertices corresponding to C in the form $(i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, where $(i_1, \ldots, i_n)$ is a permutation

of $\{1,\ldots,n\}$ and where the $P_1,\ldots,P_n$ are (possibly empty) sequences on $\{1,\ldots,n\}$. Then $(i_1,\ldots,i_n,i_1)$ is a tour π satisfying

$$w(\pi) = \sum_{j=1}^{n} w_{i_j i_{j+1}} \leq w(E) \quad (\text{where } i_{n+1} = i_1),$$

since the sum of the weights of all edges in a path from x to y is always an upper bound for w_{xy}[6] and since each edge occurs exactly once in the Euler tour C. □

We now construct spanning Eulerian multigraphs of small weight and use these to design approximative algorithms for the metric TSP. The easiest method is simply to double the edges of a minimal spanning tree, which results in the following well-known algorithm.

Algorithm 15.4.3 (tree algorithm). Let $W = (w_{ij})$ be the weight matrix for a ΔTSP on K_n.

(1) Determine a minimal spanning tree T for K_n (with respect to the weights given by W).
(2) Let $G = (V, E)$ be the multigraph which results from replacing each edge of T with two parallel edges.
(3) Determine an Euler tour C for G.
(4) Choose a tour contained in C (as described in the proof of Lemma 15.4.2).

Theorem 15.4.4. *Algorithm 15.4.3 is a 1-approximative algorithm of complexity $O(n^2)$ for ΔTSP.*

Proof. Using the algorithm of Prim, step (1) has complexity $O(n^2)$; see Theorem 4.4.4. The procedure EULER developed in Chapter 2 can be used to perform step (3) in $O(|E|) = O(n)$ steps. Clearly, steps (2) and (4) also have complexity $O(n)$. This establishes the desired complexity bound.

By Lemma 15.4.2, the tree algorithm constructs a tour π with weight $w(\pi) \leq 2w(T)$. On the other hand, the MST relaxation of Section 15.2 shows that all tours have weight at least $w(T)$. Hence $w(\pi)$ is indeed at most twice the weight of an optimal tour. □

Example 15.4.5. Let us again consider Example 15.1.2. We saw in Example 15.2.3 that the MST relaxation yields the minimal spanning tree T of weight $w(T) = 186$ displayed in Figure 15.2. A possible Euler tour for the doubled tree is

$$(Aa, Du, Ha, Be, Ha, Du, Fr, St, Ba, St, Nu, Mu, Nu, St, Fr, Du, Aa),$$

[6] Note that this is the one point in the proof where we make use of the triangle inequality.

which contains the tour

$$\pi: \; Aa — Du — Ha — Be — Fr — St — Ba — Nu — Mu — Aa$$

of length 307; see Figure 15.7. Note that Theorem 15.4.4 only guarantees that we will be able to find a tour of length ≤ 372; it is just good luck that π is actually a considerably better solution.

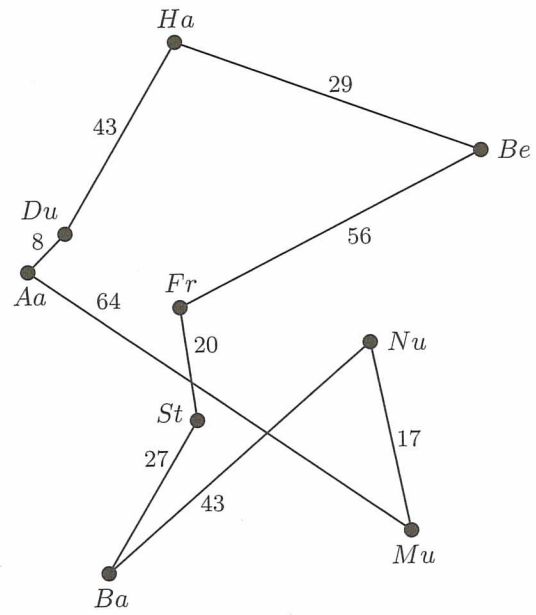

Fig. 15.7. Tour constructed by Algorithm 14.4.3

It is quite possible that Algorithm 15.4.3 constructs a tour whose weight is close to $2w(\text{TSP})$; see [LaLRS85, Chapter 5]. In contrast, the difference between the length of the tour of Example 15.4.5 and the optimal tour of Example 15.3.2 is less than 23%.

Next we present a $\frac{1}{2}$-approximative algorithm, which is due to Christofides [Chr76]; his method is a little more involved.

Algorithm 15.4.6 (Christofides' algorithm). Let $W = (w_{ij})$ be a weight matrix for a ΔTSP on K_n.

(1) Determine a minimal spanning tree T of K_n (with respect to W).
(2) Let X be the set of all vertices which have odd degree in T.
(3) Let H be the complete graph on X (with respect to the weights given by the relevant entries of W).
(4) Determine a perfect matching M of minimal weight in H.