4.5 TRAVELING SALESMAN AND RELATED PROBLEMS

627min 2–25–2003

Gregory Gutin, Royal Holloway, University of London

INTRODUCTION

The Traveling Salesman Problem (TSP) is perhaps the most studied discrete optimization problem. Its popularity is due to the facts that TSP is easy to formulate, difficult to solve, and has a large number of applications. TSP has a number of variations and generalizations extensively studied in the literature [Pu02]. In this section, we consider TSP, the Generalized TSP and the Vehicle Routing Problem.

4.5.1 Traveling Salesman Problem

K. Menger [Me32] was perhaps the first researcher to consider the Traveling Salesman Problem (TSP). He observed that the problem can be solved by examining all permutations one by one. Realizing that the complete enumeration of all permutations was not possible for graphs with a large number of vertices, he looked at the most natural *nearest neighbor* strategy and pointed out that this *heuristic*, in general, does not produce the shortest route. In fact, as we will see below, the nearest neighbor heuristic will generate the worst possible route for some problem instances. (For an interesting overview of TSP history, see [HoWo85].)

In applications, both the symmetric and asymmetric versions of the TSP are important.

Symmetric and Asymmetric TSP

DEFINITIONS

D1: ***Symmetric TSP* (*STSP*)**:
Given a complete (undirected) graph K_n with weights on the edges, find a hamiltonian cycle in K_n of minimum (total) weight.

D2: ***Asymmetric TSP* (*ATSP*)**:
Given a complete directed graph $\overleftrightarrow{K}_n$ with weights on the arcs, find a hamiltonian cycle

in $\overset{\leftrightarrow}{K}_n$ of minimum weight.

D3: The ***Euclidean TSP*** is the special case of STSP in which the vertices are points in the Euclidean plane and the weight on each edge is the Euclidean distance between its endpoints.

D4: A hamiltonian cycle in K_n or $\overset{\leftrightarrow}{K}_n$ is called a ***tour***.

NOTATION: Throughout this section, the set $\{1, 2, \ldots, n\}$ denotes the vertices of K_n or $\overset{\leftrightarrow}{K}_n$ or any other n-vertex graph under discussion.

NOTATION: By ***TSP*** we refer to both STSP and ATSP simultaneously.

Matrix Representation of TSP

Every instance of TSP can be associated with the matrix of edge-weights of the corresponding complete graph. Such a matrix is symmetric for STSP and, in general, asymmetric for ATSP.

D5: The ***distance*** (or ***weight***) ***matrix*** of an instance of STSP is the matrix $D = [d_{ij}]$, where d_{ij} is the weight of the edge between vertices i and j. The ***distance matrix*** of an instance of ATSP is the matrix $D = [d_{ij}]$, where d_{ij} is the weight of the arc directed from i to j. Accordingly, the diagonal entries d_{ii} are set to zero.

D6: An instance of TSP is said to satisfy the ***triangle inequality*** if $d_{ij} + d_{jk} \geq d_{ik}$ for all distinct vertices i, j, k.

EXAMPLES

E1: An instance of ATSP with distance matrix

$$D = \begin{pmatrix} 0 & 6 & 5 & 10 \\ 3 & 0 & 3 & 9 \\ 7 & 4 & 0 & 8 \\ 12 & 7 & 5 & 0 \end{pmatrix}$$

is shown in Figure 4.5.1. There are $3! = 6$ tours of total weight 29, 27, 30, 23, 27 and 22. The optimal tour of weight 22 is $\langle 1, 4, 3, 2, 1 \rangle$.

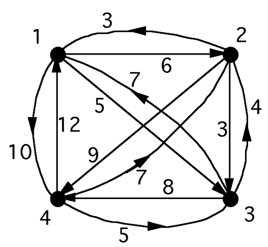

Figure 4.5.1 **An instance of ASTP.**

E2: An instance of STSP with distance matrix

$$D = \begin{pmatrix} 0 & 10 & 7 & 7 & 11 \\ 10 & 0 & 9 & 6 & 5 \\ 7 & 9 & 0 & 9 & 10 \\ 7 & 6 & 9 & 0 & 6 \\ 11 & 5 & 10 & 6 & 0 \end{pmatrix}$$

is shown in Figure 4.5.2. Since this graph has 5 vertices, there are $4!/2 = 12$ tours. The optimal tour is $\langle 1,3,2,5,4,1\rangle$ of weight 34.

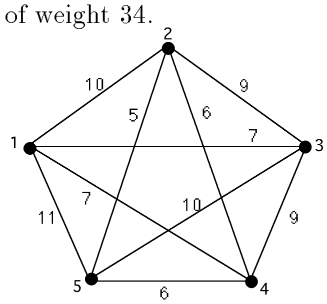

Figure 4.5.2 **An instance of SSTP.**

Algorithmic Complexity

FACTS

F1: The hamiltonian cycle problem on an n-vertex graph G can be transformed into STSP by converting G to an edge-weighted K_n as follows: assign weight 0 to each edge of G; and assign weight 1 to each edge in the edge-complement of G. A similar transformation can be used for digraphs and ATSP.

F2: Fact 1 implies that TSP is NP-hard, even if the triangle inequality holds.

By replacing the weights 0 by 1 and the weights 1 by nr in the transformation used in Fact 1, we obtain the following result.

F3: [SaGo76] For an arbitrary constant r, unless P=NP, there is no polynomial time algorithm that always produces a tour of total weight at most r times the optimal.

Exact and Approximate Algorithms

DEFINITIONS

D7: An ***exact algorithm*** is an algorithm that always produces an optimal solution.

D8: An ***approximate*** (or ***approximation***) ***algorithm*** is an algorithm that typically makes use of heuristics in reducing its computation but produces solutions that are not necessarily optimal.

E. Zemel [Ze81] defined various measures of the quality of approximate solutions, one of which is given by the Definition 9.

NOTATION: Let $\mathcal{A}$ be an approximate algorithm for TSP and I a problem instance. Then $w_{\min}(I)$, $w_{\max}(I)$, $w_{\mathcal{A}}(I)$ denote the weights, respectively, of an optimal tour, a heaviest tour, and a tour produced by algorithm $\mathcal{A}$ for instance I.

D9: The ***Zemel measure*** of algorithm $\mathcal{A}$, denoted $\rho_z(\mathcal{A})$, is the supremum of $(w_{\mathcal{A}}(I) - w_{\min}(I))/(w_{\max}(I) - w_{\min}(I))$, taken over all TSP instances I for which $w_{\max}(I) \neq w_{\min}(I)$.

FACT

F4: [HaKh01] There is a polynomial time approximate algorithm $\mathcal{A}$ for ATSP with $\rho_z(\mathcal{A}) \leq \frac{1}{2}$, and one for STSP with $\rho_z(\mathcal{A}) \leq \frac{1}{3}$.

The Euclidean TSP

Despite Fact 5 below, there was a feeling among some researchers that the Euclidean TSP is somewhat simpler than the general STSP. This was confirmed by Arora [Ar98] in 1996 (see Fact 6). Mitchell [Mi99] independently made a similar discovery a few months later (see [Ar02]).

FACTS

F5: [Pa77,GaGrJo76] Euclidean TSP is NP-hard.

F6: [Ar02] For every $\epsilon > 0$, there is a polynomial time algorithm $\mathcal{A}_\epsilon$ that, for any instance of the Euclidean TSP, finds a tour at most $1 + \epsilon$ times longer than the optimal one.

F7: As of this writing, the fastest algorithm $\mathcal{A}_\epsilon$ has time complexity $O(n \log n + n/\text{poly}(\epsilon))$ [RaSm98].

COMPUTATIONAL NOTE: These $\mathcal{A}_\epsilon$ algorithms have been implemented, but, in their current form, they are not competitive with other TSP heuristics [Ar02].

Arora's result can be generalized to d-dimensional Euclidean space for any constant d. However, Fact 8 below limits the scope of this generalization.

F8: [Tr97] There exists a constant $r > 1$ such that, for the Euclidean TSP in $O(\log n)$-dimensional Euclidean space, the problem of finding a tour that is at most r times longer than the optimal tour is NP-hard.

TSP Heuristics

Exact algorithms cannot be relied upon for applications requiring very fast solutions (online, for example) or ones that involve huge problem instances. Although approximate algorithms forfeit the guarantee of optimality, with good heuristics they can normally produce solutions close to optimal. For many applications this is good enough, since often the data are inexact anyway.

TSP heuristics can be roughly partitioned into two classes: *construction heuristics*, discussed in subsection 4.5.3, and *improvement heuristics*, discussed in subsection 4.5.4.

More comprehensive overviews of TSP heuristics can be found in [GoSt85], [JoGuMcYeZhZv02], and [JoMc02].

4.5.2 Exact Algorithms

The NP-Hardness results mentioned in the previous subsection indicate that it is rather difficult to solve large instances of TSP to optimality. Nevertheless, there are computer codes that can solve many instances with thousands of vertices within days (on a single-processor computer) [ApBiChCo98]. For discussion of TSP software implementing both exact algorithms and heuristics, see [LoPu02].

FACT

F9: The *brute-force* method of explicitly examining all possible TSP tours is impractible for even moderately sized problem instances because there are $\frac{(n-1)!}{2}$ different tours in K_n and $(n-1)!$ different tours in $\overleftrightarrow{K}_n$.

Integer Programming Approaches

Various methods have been suggested to solve TSP to optimality. They include *Lagrangian relaxation* ([BeLu00]), *dynamic programming* ([PaSt82]), and *branch-and-bound* and *branch-and-cut* (see [BaTo85],[FiLoTo02], and [Na02]). These are all well-known methods in integer programming ([Wo98]).

The earliest (and still useful) integer programming formulation of ATSP is due to Dantzig, Fulkerson and Johnson [DaFuJo54].

DANTZIG, FULKERSON AND JOHNSON FORMULATION: Define zero-one variables x_{ij} by

$$x_{ij} = \begin{cases} 1, & \text{if the tour traverses arc } (i,j) \\ 0, & \text{otherwise} \end{cases}$$

Let d_{ij} be the weight on arc (i,j). Then ATSP can be expressed as:

$$\begin{aligned} \min z = & \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \\ \text{subject to} & \sum_{i=1}^{n} x_{ij} = 1, \ j = 1,2,\ldots,n \\ & \sum_{j=1}^{n} x_{ij} = 1, \ i = 1,2,\ldots,n \\ & \sum_{i \in S} \sum_{j \in S} x_{ij} \le |S| - 1 \text{ for all } 0 < |S| < n \\ & x_{ij} = 0 \text{ or } 1, \quad i,j = 1,\ldots,n \end{aligned}$$

FACTS

F10: The first set of constraints ensures that a tour must come into vertex j exactly once, and the second set of constraints indicates that a tour must leave every vertex i exactly once. These two sets of constraints ensure that there are two arcs adjacent to each vertex, one in and one out. However, this does not prevent non-hamiltonian cycles. Instead of having one tour, the solution can consist of two of more vertex-disjoint cycles (called *sub-tours*).

F11: The third set of constraints, called *sub-tour elimination constraints*, requires that no proper subset of vertices, S, can have a total of $|S|$ arcs.

F12: The formulation without the third set of constraints is an integer programming formulation of the *Assignment Problem* that can be solved in time $O(n^3)$ [Wo98]. A solution of the Assignment Problem is a minimum-weight collection of vertex-disjoint cycles $C_1, \ldots, C_t$ spanning $\overleftrightarrow{K}_n$. If $t = 1$, then an optimal solution of ATSP has been obtained. Otherwise, one can consider two or more subproblems. For example, for a particular arc $a \in C_i$, one subproblem could add the constraint that arc a be in the solution, and a second subproblem could require that a not be in the solution. This simple idea gives a basis for branch-and-bound algorithms for ATSP.

4.5.3 Construction Heuristics

Approximate algorithms based on construction heuristics build a tour from scratch and stop when one is produced.

Greedy-Type Algorithms

The simplest and most obvious construction heuristic is *nearest neighbor.* The nearest neighbor greedy algorithm constructs a tour by always choosing as the next vertex to visit, one that is nearest to the last one visited.

Algorithm 4.5.1: Nearest Neighbor (NN)

Input: $n \times n$ distance matrix $[d_{ij}]$ and a fixed vertex i_1.
Output: TSP tour $\langle i_1, i_2, \ldots, i_n, i_1 \rangle$.

Initialize $S := \{1, 2, \ldots, n\} - \{i_1\}$.
For $k = 2, 3, \ldots, n$
  Choose i_k such that $d_{i_{k-1}, i_k} = \min_{s \in S}\{d_{i_{k-1}, s}\}$.
  $S := S - \{i_k\}$.

A second greedy-type algorithm is based on the observation that a vertex-disjoint collection of paths in $\overleftrightarrow{K}_n$ (K_n) can be extended to a tour in $\overleftrightarrow{K}_n$ (K_n).

Algorithm 4.5.2: Greedy Heuristic (GR)

Input: $n \times n$ distance matrix $[d_{ij}]$.
Output: ATSP (STSP) tour as a set S of arcs (edges).

Set $S = \emptyset$ and $m = n(n-1)$ (for ATSP) or $m = n(n-1)/2$ (for STSP).
Sort the arcs (edges) $a_1, a_2, \ldots, a_m$ in non-decreasing order of weight.
For $i = 1, 2, \ldots, m$
  If $S \cup \{a_i\}$ is the arc (edge) set of a collection of vertex-disjoint paths or is the arc (edge) set of a tour,
    $S := S \cup \{a_i\}$

EXAMPLE AND REMARK

E3: We apply *NN* to the instance of ATSP from Example 1, whose digraph is repeated in Figure 4.5.3.

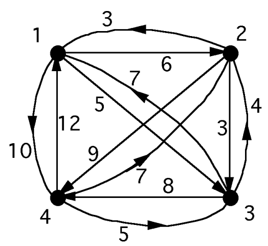

Figure 4.5.3 An instance of ASTP.

Starting from vertex 1, Algorithm *NN* moves to vertex 3, then to vertex 2, and on to vertex 4. The resulting tour is $\langle 1, 3, 2, 4, 1 \rangle$ of weight 30 (which is the worst tour).

COMPUTATIONAL NOTE: Computational experiments in [JoGuMcYeZhZv02] indicate that, in fact, on most real-world problem instances of ATSP, *NN* performs better than *GR*; *GR* fails completely on one family of instances, where the average *GR*-tour is more than 2000 % above the optimum. Computational experiments for STSP in [JoMc02] show that both *GR* and *NN* perform relatively well on Euclidean instances and perform poorly for general STSP. *GR* appears to perform better than *NN* for STSP.

Insertion Algorithms

Vertex insertion, another type of construction heuristic, applies to both Symmetric and Asymmetric TSP. For ATSP, the insertion algorithm begins with a cycle of length 2, and in each iteration, inserts a new vertex into the cycle. For STSP, the algorithm begins with a cycle of length 3. The algorithm descriptions and examples that follow are for ATSP, but with the obvious adjustments, they apply equally well to STSP.

DEFINITION

D10: Let $C = \langle i_1, i_2, \ldots, i_m, i_1 \rangle$ be the vertex sequence of a cycle in $\overset{\leftrightarrow}{K}_n$, and let v be a vertex not on C. For any arc (a, b) on cycle C, the ***insertion of vertex*** v ***at arc*** (a, b) is the operation of replacing arc (a, b) with the arcs (a, v) and (v, b) (see Figure 4.5.4). The resulting cycle is denoted $C(a, v, b)$. Thus, if $(a, b) = (i_k, i_{k+1})$, $1 \leq k \leq m-1$, then $C(a, v, b) = \langle i_1, i_2, \ldots, a, v, b, \ldots i_m, i_1 \rangle$, and if $(a, b) = (i_m, i_1)$, then $C(a, v, b) = \langle i_1, i_2, \ldots i_{m-1}, a, v, b \rangle$.

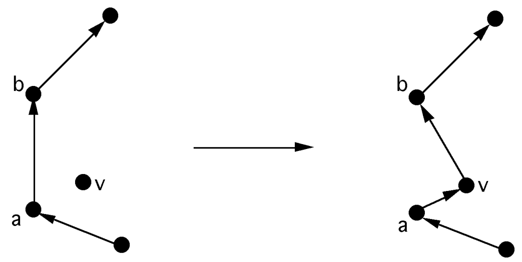

Figure 4.5.4 **Insertion of vertex v at arc (a, b).**

Algorithm 4.5.3: Vertex Insertion (VI)

Input: $n \times n$ distance matrix $[d_{ij}]$.
Output: TSP tour $\langle i_1, i_2, \ldots, i_n, i_1 \rangle$.

Let i_1 and i_2 be two vertices of $\overset{\leftrightarrow}{K}_n$, chosen by some rule.
Initialize cycle $C = \langle i_1, i_2, i_1 \rangle$.
For $s = 3, \ldots, n$
  Let i_s be a vertex not on cycle C, chosen by some rule [*].
  Insert vertex i_s at an arc (a^*, b^*) of cycle $C = \langle i_1, i_2, \ldots, i_{s-1}, i_1 \rangle$
   such that the weight of $C(a^*, i_s, b^*)$ is minimum among the
   cycles $C(a, i_s, b)$ for all arcs (a, b) in C.
  $C := C(a^*, i_s, b^*)$.

REMARK

R1: *Random vertex insertion*, *nearest vertex insertion*, and *farthest vertex insertion*, which are defined below, are three different versions of algorithm *VI*. Each is determined by how rule [*] chooses vertex i_s.

NOTATION: Given a vertex v and a cycle C in $\overleftrightarrow{K}_n$, $d(v, C)$ denotes the distance from v to C, that is, $d(v, C) = \min_{x \in V(C)} \{d_{vx}\}$.

DEFINITIONS

D11: The ***random vertex insertion (RVI)*** chooses vertex i_s randomly.

D12: The ***nearest vertex insertion (NVI)*** chooses vertex i_s so that its distance to cycle C is a minimum. That is, $d(i_s, C) = \min_{v \notin V(C)} \{d(v, C)\}$.

D13: The ***farthest vertex insertion (FVI)*** chooses vertex i_s so that its distance to cycle C is a maximum. That is, $d(i_s, C) = \max_{v \notin V(C)} \{d(v, C)\}$.

COMPUTATIONAL NOTE: The vertex insertion heuristics described above perform quite well for Euclidean TSP (see [JoMc02]). Computational experiments with *RVI* for ATSP in [GlGuYeZv01] show that *RVI* is good only for instances close to Euclidean.

Minimum Spanning Tree Heuristics

There are many more construction heuristics for TSP and especially for STSP, see [JoGuMcYeZhZv02,JoMc02]. For STSP, the heuristics that are often given in the literature include the *Double Minimum Spanning Tree (DMST)* and *Christofides* heuristics (see, e.g., Algorithms 6.4.2 and 6.4.3 in [GrYe99]). We will describe only the Christofides heuristic as it is significantly better than *DMST* from both theoretical and experimental point of view.

Algorithm 4.5.4: Christofides Heuristic (CH)

Input: $n \times n$ distance matrix $[d_{ij}]$.
Output: STSP tour $\langle i_1, i_2, \ldots, i_n, i_1 \rangle$.

```
Find a minimum spanning tree T in K_n.
Find a minimum-weight perfect matching M in the subgraph of K_n
  induced on the odd-degree vertices of T.
Create an eulerian subgraph H of K_n with edge set E(H) = E(T) ∪ M.
Construct an eulerian tour W of H.
Let W = ⟨i_1, i_2, ..., i_m, i_1⟩ (written as a sequence of vertices).
For s = 3, ..., n
    If i_s = i_t for some t < s, then delete i_{t+1}, ..., i_s from W.
```

F13: [JoPa85] Algorithm *CH* can be implemented to run in time $O(n^3)$.

D14: The operations to eliminate repeated vertices in the eulerian tour W in order to obtain an STSP tour are called ***shortcuts***.

COMPUTATIONAL NOTE: Implementing the ordinary shortcuts described in Algorithm *CH* already produces a relatively good heuristic for the Euclidean TSP [JoMc02]. However, so-called greedy shortcuts (see [JoMc02]) result in a modification of *CH*, which seems to be one of the best construction heuristics for the Euclidean TSP.

Worst Case Analysis of Heuristics

While computational experiments are important in the evaluation of heuristics, they cannot cover all possible families of instances of TSP and, in particular, they normally do not cover the most difficult instances. Moreover, certain applications may produce families of instances that are much harder than those normally used in computational experiments. For example, such instances can arise when Generalized TSP (discussed in subsection 4.5.5) is transformed into TSP. Thus, theoretical analysis of the worst possible cases is also important in evaluating and comparing TSP heuristics.

FACTS

F14: [GuYeZv02-a] For every $n \geq 3$, there is an instance of ATSP and an instance of STSP with n vertices satisfying the triangle inequality on which both *NN* and *GR* compute the unique worst possible tour.

F15: [Ru73] Let H be a tour produced by *RVI* for an instance I_n of STSP with $n \geq 3$ vertices. Then H is not worse than at least $(n-2)!$ tours when n is odd and $(n-2)!/2$ tours when n is even (including H itself).

F16: [GuYeZv02-a] For every $n \geq 2$, $n \neq 6$ and every instance of ATSP with n vertices, *RVI* computes a tour T that is not worse than at least $(n-2)!$ tours including T itself.

F17: For STSP with triangle inequality, the *DMST* algorithm always produces a tour no more than twice as long as the optimal one, while *HC* produces tours never worse than 1.5 times the optimum (see [JoPa85]). However, there are instances for which *DMST* produces the unique worst possible tour, and there are instances for which *CH* produces a tour worse than all but at most $\lceil n/2 \rceil!$ tours [PuMaKa].

REMARKS

R2: A simplified proof of Fact 14 can be found in [GuYeZv02-b]. It is based on a proof of a much more general result for the greedy algorithm in combinatorial optimization (see [GuYe02-a]).

R3: A proof of Fact 16 is similar to the proof of Fact 15, but is based on a different result that was first proved for n odd by V. I. Sarvanov [Sa76], and for n even by Gutin and Yeo [GuYe02-b]. The proofs use decompositions of K_n and $\overset{\leftrightarrow}{K}_n$, into hamiltonian cycles, which exist for K_n if and only if n is odd (see, e.g., [Ha69]), and for $\overset{\leftrightarrow}{K}_n$ if and only if $n \neq 4$ or 6 [Ti80].

4.5.4 Improvement Heuristics

Approximate algorithms based on improvement heuristics start from a tour (normally obtained using a construction heuristic) and iteratively improve it by changing some parts of it at each iteration. The best known tour improvement procedures are based on *edge exchange*, in which a tour is improved by replacing k its edges (arcs) with k edges (arcs) not in the solution.

COMPUTATIONAL NOTE: For many combinatorial optimization problems, well-known metaheuristics including *tabu search*, *simulated annealing* and *genetic algorithms* provide the best tools for producing good quality approximate solutions. This has not been the case for TSP, for which variations of the edge-exchange algorithms of Lin and Kernighan (*Lin-Kernighan local search*) are still state-of-the-art ([JoGuMcYeZhZv02], [JoMc02]). They are typically much faster than the exact algorithms, yet often produce solutions very close to the optimal one. Interested readers can find a detailed description of the Lin-Kernighan local search and its generalizations in [ReGl02]. Although the Lin-Kernighan local search can be applied only to STSP, ATSP can be transformed into STSP (see, e.g., [JoGuMcYeZhZv02]).

DEFINITIONS

D15: For STSP, the ***2-opt*** algorithm starts from an initial tour T and tries to improve T by replacing two of its non-adjacent edges with two other edges to form another tour (see Figure 4.5.5). Once an improvement is obtained, it becomes the new T. The procedure is repeated as long as an improvement is possible (or a time limit is exceeded).

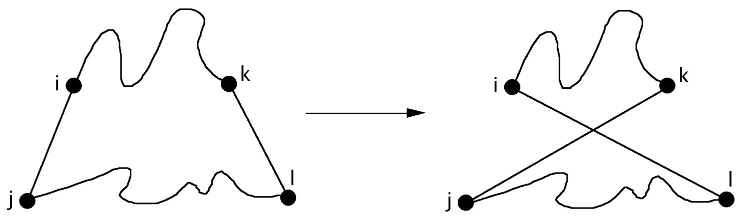

Figure 4.5.5 **Edges $\{i,j\}$ and $\{k,l\}$ are replaced by $\{i,l\}$ and $\{j,k\}$.**

D16: For $k \geq 3$, the ***k-opt*** algorithm is the same as 2-opt except that k edges (arcs) are replaced at each iteration.

D17: The ***best improvement k-opt*** is similar to the k-opt defined above; the difference is that every set of k arcs (edges) is tried for deletion from T and all possibilities of adding k arcs (edges) are considered before the *best* replacement of T is retained (as a replacement for T). The procedure is repeated.

FACTS

F18: [PuMaKa] Although best improvement 2-opt can take exponential time to find a local optimum, any (possibly sub-optimal) tour obtained after $O(n^3 \log n)$ time is not worse than $\frac{1}{n-1}$ of all STSP tours. Similar statements hold for the best improvement k-opt, $k \geq 3$.

Exponential Neighborhoods

Best improvement k-opt considers $\Theta(n^k)$ tours that can be obtained from a tour T by replacing edge-exchanges involving exactly k edges (arcs). Thus, it considers only a polynomial number of tours in the *neighborhood* of T. For TSP, one can construct various neighborhoods with an exponential number of tours in which the best tour can be found in polynomial time. In particular, there exist TSP neighborhoods of size $2^{\Theta(n \log n)}$ ($2^{\Theta(n)}$), where the best tour can be found in time $O(n^2)$ ($O(n)$). These neighborhoods are discussed in [AhErOrPu02], [DeWo00], and [GuYeZv02-b].

COMPUTATIONAL NOTE: While having seemingly strong theoretical properties, TSP exponential-neighborhood, local-search algorithms have not shown strong computation properties so far. Perhaps, this is due to the fact that it is not the size of the neighborhood that matters, but the total number of tours of TSP that are worse than the best tour in the neighborhood. This may explain why computational experiments show that some exponential-neighborhood, local-search heuristics are worse than the (seemingly much weaker) 3-opt.

4.5.5 The Generalized TSP

The *Generalized TSP* has numerous applications and is one of the most studied extensions of TSP [FiSaTo02].

DEFINITIONS

D18: The ***Generalized Asymmetric Traveling Salesman Problem (GATSP)***: Given a weighted complete digraph $\overleftrightarrow{K}_n$ and a partition $V_1, \ldots, V_k$ of its vertices, find a minimum-weight cycle containing exactly one (at least one) vertex from each set V_i, $i = 1, \ldots, k$.

D19: The sets V_i are called ***clusters***, and a cycle containing exactly one (at least one) vertex from each cluster is called a ***tour***.

D20: The ***Generalized Symmetric Traveling Salesman Problem (GSTSP)*** is formulated similarly with $\overleftrightarrow{K}_n$ replaced by K_n.

REMARK

R4: Observe that the requirements 'at least one' and 'exactly one' in GATSP and GSTSP coincide when the triangle inequality holds. The 'exactly one' versions of GATSP and GSTSP have received the most attention in the literature, and only these versions are discussed here.

Transforming Generalized TSP to TSP

One of the ways to solve instances of the Generalized TSP is to transform them into TSP instances. The most efficient transformations from GATSP to ATSP and from GSTSP to STSP appear to be the ones given in [NoBe93] and [LaSe99], respectively.

FACTS

F19: In the transformation of [NoBe93], from GATSP into ATSP, the number of vertices remains the same. Weights are modified so that an optimal ATSP tour must visit all the vertices that belong to the same cluster in the original problem before moving on to the next cluster. This is achieved by adding a large positive constant M to the weight of each inter-cluster arc. If the constant is large enough, an optimal tour will contain exactly k such heavy arcs, thus ensuring that no cluster is visited more than once.

F20: In the transformation of [LaSe99], from a GSTSP instance into an STSP instance, we first add a sufficiently large positive constant to every edge-weight, if needed, to ensure that all edge-weights are nonnegative. Then we consider each cluster V_i of cardinality at least 2. For each vertex v_i in such a cluster, we create a copy v_i'. In each such cluster, we form a hamiltonian cycle $C = \langle v_1, v_1', \ldots, v_t, v_t', v_1 \rangle$ and assign weight $-M$ to every edge of the form $v_i v_i'$ and weight $-2M$ to the rest of the edges in cycle C, where M is any constant larger than the sum of n heaviest edges in the GSTSP instance. The weights of the remaining edges within the clusters and between the clusters are inherited from the corresponding weights of the GSTSP instance. Clearly, an optimal tour T of the resulting STSP instance will use all edges of weight $-2M$, all but one of the $(-M)$-weight edges from each cluster of cardinality at least 2, and edges between the clusters. By contracting every vertex v_i and its copy v_i' in T, we obtain an optimal tour of the GSTSP instance.

F21: For the transformations in [LaSe99] and [NoBe93], there is a bijection between *optimal* tours in the original problem and those in the transformed one, making the transformations suitable for exact algorithms.

Exact Algorithms

FACTS

F22: Computational experiments ([BeGuPeYeZv] and [LaSe99]) have shown that the transformations given in [NoBe93] and [LaSe99] can be used to solve to optimality small to moderate instances of Generalized TSP. However, even small instances require substantial computation because of the corresponding TSP instances' very large positive (negative) weights on some of its arcs (edges).

F23: A successful branch-and-cut algorithm for GSTSP is described and analyzed in [FiSaTo02], and a Lagrangian-based approach for GATSP is given in [NoBe91].

The next result appears to be a major stumbling block for using a standard branch-and-bound for ATSP adapted for GATSP.

F24: [GuYe] Let $D = (V, A)$ be a digraph and let $V_1, V_2, \ldots, V_k$ be a partition of V. The problem of checking whether D has 1-regular subdigraph containing exactly one vertex from each $V_1, V_2, \ldots, V_k$ is NP-complete even if $|V_i| \leq 2$ for every $i = 1, 2, \ldots, k$. (A digraph H is 1-regular if the indegree and outdegree of every vertex in H equal 1.)

Approximate Algorithms

Although several TSP heuristics easily extend to the Generalized TSP, it seems better to first transform a Generalized TSP instance into a TSP instance, and then use heuristics on the TSP instance.

F25: For the transformations in [LaSe99] and [NoBe93], not every tour in the transformed problem corresponds to a tour in the original problem, making the transformations less suitable for approximate algorithms.

F26: Two other transformations, described in detail in [FiSaTo02] and [BeGuPeYeZv], are used in producing approximate (or exact) solutions. They apply to both GATSP and GSTSP. The description below is for an instance of GATSP with k clusters, $V_1, V_2, \ldots, V_k$.

TRANSFORMATION:

(1) Let G be a complete directed graph on k vertices, $v_1, \ldots, v_k$, such that the weight of arc (v_i, v_j) in G equals some function ϕ of the weights of the arcs in $\overset{\leftrightarrow}{K}_n$ from cluster V_i to cluster V_j. (In [FiSaTo02], the function ϕ is the minimum of these arc-weights, and in [BeGuPeYeZv], ϕ is the average weight.)

(2) Let $\langle v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1}\rangle$ be an approximate (or exact) solution for the ATSP instance on G.

(3) Find a minimum-weight cycle C having exactly one vertex from each cluster and traversing the clusters in the order $V_{i_1}, V_{i_2}, \ldots, V_{i_k}$ by solving shortest path problems in $|V_1|$ acyclic digraphs.

COMPUTATIONAL NOTE: Computational experiments for GATSP in [BeGuPeYeZv] indicate that the two different weight-functions ϕ produce solutions of comparable quality. However, theoretical analysis in [BeGuPeYeZv] shows that the average-weight function appears to have better worst-case performance than the minimum-weight one.

4.5.6 The Vehicle Routing Problem

The *Vehicle Routing Problem (VRP)* was introduced by Dantzig and Ramser [DaRa59]. This problem (including its versions with additional constraints) seems to be the most applicable of all generalizations of TSP. Vehicle routing is the generic name given to a large family of problems involving the distribution of goods, information, services or people. A particularly important special case of VRP is that of minimizing the total distance traveled by a fleet of vehicles that deliver goods ordered by customers. The vehicles are assumed to have equal capacity Q, and their delivery tours start and end at a central depot.

DEFINITIONS

D21: Given a weighted complete directed or undirected graph on vertices $\{0, 1, \ldots, n\}$, a demand $d_i \geq 0$ for $i = 1, 2, \ldots, n$, and two parameters Q and k, a ***CVRP tour*** is a collection of k cycles $C_1, C_2, \ldots, C_k$, which contain all the vertices, pairwise intersect only in vertex 0, and satisfy $\sum_{i \in V(C_j)} d_i \leq Q$ for each $j = 1, 2, \ldots, k$.

D22: The ***Capacitated Vehicle Routing Problem (CVRP)***:
Given a weighted complete directed or undirected graph on $n + 1$ vertices, a demand $d_i \geq 0$ for $i = 1, 2, \ldots, n$, and two parameters Q and k, find a CVRP tour for which the total weight of the cycles is minimum.

REMARKS

R5: Often, practitioners and researchers consider additional complicating constraints: the total weight of each cycle is limited, each customer (vertex) i must be visited within a prescribed *time window*, vehicles are allowed to have different capacities, routes of different vehicles cannot cross, etc. [PoKaWa99].

R6: In most research papers, the symmetric CVRP (on K_{n+1}) is considered. Nevertheless, the asymmetric (i.e., 'directed') CVRP version is also of interest [Vi96].

Exact Algorithms

FACTS

F27: The most efficient exact algorithms for symmetric CVRP are those based on branch-and-cut ([BlHo00], [NaRi01], [RaKoPuTr]).

F28: For the asymmetric version of CVRP it seems that the state-of-the-art exact algorithms still use branch-and-bound [ToVi01,ToVi02].

F29: Since CVRP has aspects of both TSP and *Bin Packing*, *set-covering* methods can sometimes be applied to CVRP and its generalizations with great success [BrSi01].

COMPUTATIONAL NOTE: The exact algorithms appear to be less powerful for CVRP than they are for TSP. Although they are able to solve some instances with 100 or more vertices, the exact algorithms were unable to solve an instance of symmetric CVRP with as few as 51 vertices [RaKoPuTr,ToVi02]. Often, practical versions of CVRP have various complicating constraints that cannot be tackled by exact algorithms. Thus, heuristics are of great importance for CVRP.

Heuristics for CVRP

CVRP heuristics fall roughly into two categories: those that produce a CVRP tour relatively quickly; and those that try to produce a near-optimal solution, using a substantial amount of computing if necessary. The latter kind are mostly metaheuristic-based algorithms. *Tabu search* seems to provide a good tradeoff between the quality of solution and running time ([ErOrSt], [GeLaPo01], [Ta93], [ToVi98]).

Fast CVRP heuristics are of great importance, supplying quick and flexible solutions, good starting tours for metaheurisic-based algorithms, and upper bounds for exact algorithms. We close this section with brief descriptions of three classes of fast CVRP heuristics: *savings heuristics*, *insertion heuristics*, and *two-phase heuristics*.

REMARK

R7: The descriptions that follow are for the asymmetric CVRP, but they also apply to the symmetric CVRP with 'digraph' replaced by 'graph' and 'arc' replaced by 'edge.'

NOTATION: For a vertex subset S, $t(S)$ denotes (an approximation of) the weight of an optimal TSP tour of the induced subdigraph on S.

NOTATION: The total demand of a vertex subset S is $d(S) = \sum_{i \in S} d_i$.

Savings Heuristics

The Clarke-Wright savings heuristic is perhaps the earliest [ClWr64] and best known heuristic for the VRP. Here, we describe a generic savings heuristic, whose concrete implementations may be found in [AlGa91], [ClWr64], and [LaSe01].

DEFINITIONS

D23: A ***merge*** of cycles C_1 and C_2, denoted merge(C_1, C_2), is a cycle whose vertex set equals $V(C_1) \cup V(C_2)$. The resulting cycle is determined by some prescribed rule. Cycles C_1 and C_2 can be merged only if the total demand of their vertices does not exceed capacity Q (i.e., $d(V(C_1) \cup V(C_2)) \leq Q$).

D24: Given cycles C_1 and C_2, the ***saving*** of merge(C_1, C_2), denoted $s(C_1, C_2)$, is given by $s(C_1, C_2) = t(V(C_1)) + t(V(C_2)) - t(V(C_1) \cup V(C_2))$.

D25: Let $R = C_1, C_2, \ldots, C_m$ be a collection of m cycles of $\overset{\leftrightarrow}{K}_n$ whose pairwise intersections are vertex 0. The ***savings digraph***, $D(R)$, is the weighted digraph on m vertices, labeled $C_1, C_2, \ldots, C_m$, such that arc (C_i, C_j) exists if $d(V(C_1) \cup V(C_2)) \leq Q$, and the weight assigned to arc (C_i, C_j) is the saving $s(C_i, C_j)$.

REMARKS

R8: In the *Clarke-Wright savings algorithm* ([ClWr64],[LaSe01]), the weight of cycle C is used as an estimate of $t(V(C))$. To obtain the exact value of $t(V(C))$, one would have to solve a TSP on the induced subdigraph on $V(C)$, which may be too costly computationally.

R9: The simplest way to merge cycles C_1 and C_2 is the one used in the Clarke-Wright algorithm. If $(i, 0)$ is the arc in C_1 that enters vertex 0, and $(0, j)$ is the arc in C_2 that leaves vertex 0, then these two arcs are deleted and arc (i, j) is added to complete the new cycle. (See Figure 4.5.6.)

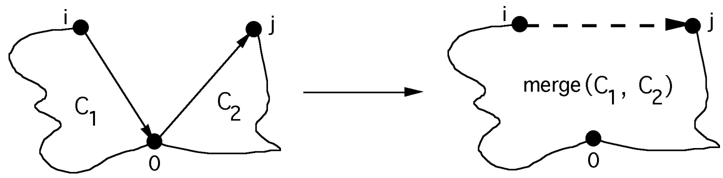

Figure 4.5.6 **A Clarke-Wright merge of cycles C_1 and C_2.**

Algorithm 4.5.5: Savings Heuristic (SH)

Input: distance matrix $[d_{ij}]$, demands d_i, $i = 1, \ldots, n$,
capacity Q, and number of vehicles k.
Output: CVRP tour $R = \{C_1, \ldots, C_k\}$.

Initialize cycle count $m = n$ and cycles $C_i = \langle 0, i, 0 \rangle$, $i = 1, \ldots, m$.
Initialize $R = \{C_1, \ldots, C_m\}$.
While $m > k$
  Construct savings digraph $D(R)$.
  Construct a matching M in $D(R)$ with $|M| \leq m - k$.
  For each arc (C_i, C_j) in M,
    $R := (R - \{C_i, C_j\}) \cup \text{merge}(C_i, C_j)$
    $m := m - 1$

REMARKS

R10: The easiest way to construct M is to simply choose a pair (C_a, C_b) with maximum saving $s(C_a, C_b)$ as the only arc in M. In some versions in which cycle-merging occurs in parallel, M is built in a greedy manner [LaSe01].

R11: Often, after R is produced by algorithm *SH*, each of the cycles in R is improved by some TSP improvement heuristic (see Section 4.5.4). For example, in [LaSe01], a CVRP tour found by the Clarke-Wright algorithm is improved by applying the best improvement 3-opt to each of its cycles.

Insertion Heuristics

In CVRP iteration algorithms [LaSe01], we start from k cycles of the form $C_p = \langle 0, i_p, 0 \rangle$. The vertices outside of the cycles are inserted one by one in sequential or parallel manner. The word 'parallel' here means that a vertex is inserted in one of the current cycles C_p for which the insertion is most beneficial. In the sequential mode, we start constructing a new cycle only when the previous one cannot be increased because of the capacity constraints. The cost of insertion of a vertex i into a current cycle C_p

can be measured by $t(V(C) \cup \{i\}) - t(V(C))$.

REMARKS

R12: An example of such an algorithm is the Christofides-Mingozzi-Toth insertion heuristic [ChMiTo79].

R13: Fisher and Jaikumar [FiJa81] suggested inserting all vertices at the same time. They apply the *Generalized Assignment Problem* to find 'optimal' insertions that do not violate the capacity constraints.

Two-phase Heuristics

The basic idea of *two-phase heuristics* is to partition vertices $\{1, 2, \ldots, n\}$ into k clusters $V_1, \ldots, V_k$ and solve TSP for each of the graphs induced by $V_i \cup \{0\}$, $i = 1, 2, \ldots, k$. Wren and Holliday [WrHo72] suggested a sweeping technique for the Euclidean CVRP, in which the depot 0 and vertices i, $1 \leq i \leq n$ are points on the Euclidean plane, and d_{ij} is the Euclidean distance between i and j.

We introduce a polar coordinate system, in which one of the n vertices, say i, is chosen as the reference point, with polar coordinates $(0, d_{0i})$. Then every vertex $j \neq i$ has coordinates (ϕ_j, d_{0j}), where ϕ_j is the angle between the rays from 0 to i and from 0 to j.

The sweeping algorithm in its simplest form is as follows.

Algorithm 4.5.6: Sweeping Heuristic

Input: distance matrix $[d_{ij}]$, polar angles ϕ_i for $1 \leq i \leq n$,
demands d_i, $i = 1, \ldots, n$, capacity Q, and number of vehicles k.
Output: CVRP tour consisting of cycles C_j, $j = 1, \ldots, k$.

```
Sort the vertices i_1, i_2, ..., i_n such that φ_{i_s} ≤ φ_{i_{s+1}}, s = 1, ..., n − 1.
Initialize S_j = ∅, j = 1, ..., k.
Set j = 1.
For s = 1, ..., n
    If d(S_j ∪ {i_s}) > Q
        j := j + 1
    S_j := S_j ∪ {i_s}
For j = 1, ..., k
    Let C_j be a TSP tour for the subgraph induced on S_j ∪ {0}.
```

R14: An extension of this approach to the general CVRP is described in [BrSi95]. Another example of a two-phase heuristic is a truncated branch-and-bound provided in [CHMiTo79].

REFERENCES

[AhErOrPu02] R. K. Ahuja, O. Ergun, J. B. Orlin and A. P. Punnen, A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* 123 (2002), 75-102.

[AlGa91] K. Altinkemer and B. Gavish, Parallel savings based heuristics for the delively problem. *Oper. Res.* 39 (1991) 456-469.

[ApBiChCo98] D. Applegate, R. E. Bixby, V. Chvátal and W. Cook, On the Solution of Traveling Salesman Problem. *Doc. Math., J. DMV, Extra Vol. ICM Berlin 1998, III* (1998) 645-656. The 12/15/1999 release of the `Concorde` code is currently available from `http://www.math.princeton.edu/tsp/concorde.html`

[Ar98] S. Arora, Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *JACM* 45 (1998), 753-782. Preliminary version in Proc. 37th IEEE Symp. Found. Comput. Sci., 1996, and Proc. 38th IEEE Symp. Found. Comput. Sci., 1997.

[Ar02] S. Arora, Approximation algorithms for geometric TSP. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[BaTo85] E. Balas and P. Toth, Branch and Bound Methods. In *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds.), Wiley, 1985.

[BeLu00] A. Belloni and A. Lucena, A relax and cut algorithm for the traveling salesman problem. *A talk at Intern. Symp. Math. Prog., Atlanta*, 2000.

[BeGuPeYeZv] D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo and A. Zverovitch, Transformations of Generalized ATSP into ATSP: experimental and theoretical study. To appear in *Oper. Res. Lett.*

[BlHo00] U. Blasum and W. Hochstättler, Application of the brach and cut method to the vehicle routing problem. Zentrum für Angewandte Informatik, Köln, Tech. Report zrp2000-386.

[BrSi95] J. Bramel and D. Simchi-Levi, A location based heuristic for general routing problems. *Oper. Res.* 43 (1995) 649-660.

[BrSi01] J. Bramel and D. Simchi-Levi, Set-Covering Based Algorithms for the Capacitated VRP. In *The Vehicle Routing Problem* (P. Toth and D. Vigo, eds.), SIAM, 2001.

[ClWr64] G. Clarke and J. W. Wright, Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.* 12 (1964), 568-581.

[ChMiTo79] N. Christofides, A. Mingozzi and P. Toth, The Vehicle Routing Problem. In *Combinatorial Optimization* (A. Mingozzi, P. Toth and C. Sandi, eds.), Wiley, 1979.

[DaFuJo54] G. B. Dantzig, D. R. Fulkerson and S. M. Johnson, Solution of large scale traveling salesman problem. *Oper. Res.* 2 (1954), 393-410.

[DaRa59] G. B. Dantzig and R. H. Ramser, The truck dispatching problem. *Manag. Sci.* 6 (1959) 80-91.

[DeWo00] V. G. Deineko and G. J. Woeginger, A study of exponential neighbourhoods for the traveling salesman problem and the quadratic assignment problem. *Math. Program., Ser. A* 87 (2000), 519-542.

[ErOrSt] O. Ergun, J. B. Orlin and A. Steele-Feldman, Creating very large scale neighborhoods out of smaller ones by compounding moves: a study on the vehicle routing problem. Manuscript, 2002.

[FiLoTo02] M. Fischetti, A. Lodi and P. Toth, Exact Methods for the Asymmetric Traveling Salesman Problem. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[FiSaTo02] M. Fischetti, J. J. Salazar-González and P. Toth, The Generalized Traveling Salesman and Orienteering Problem. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[FiJa81] M. L. Fisher and R. Jaikumar, A generalized assignment heuristic for vehicle routing, *Networks* 11 (1981) 109-124.

[GaGrJo76] M. R. Garey, R. L. Graham and D. S. Johnson, Some NP-complete geometric problems. In *Proc. 8th ACM Symp. Theory Comput.* (1976), 10-22.

[GeLaPo01] M. Gendreau, G. Laporte and J. Y. Potvin, Metaheuristics for the Capacitated VRP. In *The Vehicle Routing Problem* (P. Toth and D. Vigo, eds.), SIAM, 2001.

[GlGuYeZv01] F. Glover, G. Gutin, A. Yeo and A. Zverovich, Construction heuristics for the asymmetric TSP. *Eur. J. Oper. Res.* 129 (2001), 555-568.

[GoSt85] B. L. Golden and W. R. Stewart, Empirical Analysis of Heuristics. In *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds.), Wiley, 1985.

[GrYe99] J. L. Gross and J. Yellen, *Graph Theory and Its Applications*, CRC Press, 1999.

[GuYe02-a] G. Gutin and A. Yeo, Anti-matroids. *Oper. Res. Lett.* 30 (2002) 97-99.

[GuYe02-b] G. Gutin and A. Yeo, Polynomial approximation algorithms for the TSP and the QAP with factorial domination number. *Discrete Appl. Math.* 119 (2002), 107-116.

[GuYe] G. Gutin and A. Yeo, Assignment Problem based algorithms are impractical for the Generalized TSP. To appear in *Ausralasian J. Combinatorics.*

[GuYeZv02-a] G. Gutin, A. Yeo and A. Zverovich, Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Appl. Math.* 117 (2002), 81-86.

[GuYeZv02-b] G. Gutin, A. Yeo and A. Zverovitch, Exponential Neighborhoods and Domination Analysis for the TSP. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[HaKh01] R. Hassin and S. Khuller, z-Approximations. *J. Algorithms* 41 (2001) 429-442.

[Ha69] F. Harary, *Graph Theory*, Addison-Wesley, 1969.

[HoWo85] A. J. Hoffman and P. Wolfe, History. In *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds.), Wiley, 1985.

[JoGuMcYeZhZv02] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang and A. Zverovitch, Experimental Analysis of Heuristics for ATSP. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, Dordrecht, 2002.

[JoMc02] D. S. Johnson and L. A. McGeoch, Experimental Analysis of Heuristics for STSP. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[JoPa85] D. S. Johnson and C. H. Papadimitriou, Performance guarantees for heuristics. In *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds.), Wiley, 1985.

[LaSe99] G. Laporte and F. Semet, Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR* 37 (1999), 114-120.

[LaSe01] G. Laporte and F. Semet, Classical Heuristics for the Capacitated VRP. In *The Vehicle Routing Problem* (P. Toth and D. Vigo, eds.), SIAM, 2001.

[LoPu02] A. Lodi and A. P. Punnen, TSP Software. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[Me32] K. Menger, Das botenproblem. *Ergebnisse Eines Mathematischen Kolloquiums* 2 (1932), 11-12.

[Mi99] J. C. B. Mitchell, Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial time approximation scheme for geometric TSP, k-MST, and related problem. *SIAM J. Comput.* 28 (1999), 1298-1309.

[Na02] D. Naddef, Polyhedral Theory and Branch-and-Cut Algorithms for the Symmetric TSP. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[NaRi01] D. Naddef and G. Rinaldi, Branch-and-Cut Algorithms for the Capacitated VRP. In *The Vehicle Routing Problem* (P. Toth and D. Vigo, eds.), SIAM, 2001.

[NoBe91] C. E. Noon and J. C. Bean, A Lagrangian based approach for the asymmetric generalized traveling salesman problem. *Oper. Res.* 39 (1991), 623-632.

[NoBe93] C. E. Noon and J. C. Bean, An efficient transformation of the generalized traveling salesman problem. *INFOR* 31 (1993), 39-44.

[Pa77] C. H. Papadimitriou, The Euclidean traveling salesman problem is NP-complete. *Theoret. Comput. Sci.* 4 (1977), 237-244.

[PaSt82] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, 1982.

[PoKaWa99] A. Poot, G. Kant and A. P. M. Wagelmans, A savings based method for real-life vehicle routing problems. Erasmus University Rotterdam, Econometric Institute Report Ei 9938/A.

[Pu02] A. P. Punnen, The Traveling Salesman Problem: Aplications, Formulations and Variations. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[PuMaKa] A. P. Punnen, F. Margot and S. Kabadi, TSP heuristics: domination analysis and complexity. To appear in *Algorithmica*.

[RaKoPuTr] T. K. Ralphs, L. Kopman, W. R. Pulletblank and L. E. Trotter, On the capacitated vehicle routing problem. To appear in *Math. Program.*

[RaSm98] S. Rao and W. Smith, Approximating geometric graphs via "spanners" and "banyans". *Proc. 30th Ann. ACM Symp. Theory Comput.* (1998), 540-550.

[ReGl02] C. Rego and F. Glover, Local Search and Metaheuristics. In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

[Ru73] V. I. Rublineckii, Estimates of the accuracy of procedures in the Traveling Salesman Problem. *Numerical Mathematics and Computer Technology* no. 4 (1973), 18-23 (in Russian).

[SaGo76] S. Sahni and T. Gonzalez, P-complete approximation problems. *JACM* 23 (1976), 555-565.

[Sa76] V. I. Sarvanov, On the minimization of a linear form on a set of all n-elements cycles. *Vestsi Akad. Navuk BSSR, Ser. Fiz.-Mat. Navuk* no. 4 (1976), 17-21 (in Russian).

[Ta93] E. Taillard, Parallel iterative search methods for vehicle routing problem. *Networks* 23 (1993), 661-673.

[Ti80] T. W. Tillson, A Hamiltonian decomposition of K^*_{2m}, $2m \geq 8$. *J. Combin. Theory, Ser. B* 29 (1980), 68-74.

[ToVi01] P. Toth and D. Vigo, Branch-and-Bound Algorithms for the Capacitated VRP. In *The Vehicle Routing Problem* (P. Toth and D. Vigo, eds.), SIAM, 2001.

[ToVi02] P. Toth and D. Vigo, Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Appl. Math.* 123 (2002), 487-512.

[Tr97] L. Trevisan, When Hamming meets Euclid: the approximability of geometric TSP and MST. *Proc. 29th ACM Symp. Theory Comput.* (1997), 21-39.

[Vi96] D. Vigo, A heuristic algorithm for the asymmetric capacitated vehicle routing problem. *Europ. J. Oper. Res.* 89 (1996), 108-126.

[Wo98] L. A. Wolsey, *Integer Programming*, Wiley, 1998.

[WrHo72] A. Wren and A. Holliday, Computer scheduling of vehicles from one or more depots to a number of delivery points. *Oper. Res. Quart.* 23 (1972) 333-344.

[Ze81] E. Zemel, Measuring the quality of approximate solutions to zero-one programming problems. *Math. Oper. Res.* 6 (1981) 319-332.

GLOSSARY

approximate (or approximation) algorithm: an algorithm that typically makes use of heuristics in reducing its computation but produces solutions that are not necessarily optimal.

Asymmetric TSP (ATSP): Given a complete directed graph $\overleftrightarrow{K}_n$ with weights on the arcs, find a minimum-weight hamiltonian cycle in $\overleftrightarrow{K}_n$.

Capacitated Vehicle Routing Problem (CVRP): Given a weighted directed or undirected complete graph on $n+1$ vertices, a demand $d_i \geq 0$ for $i = 1, 2, \ldots, n$, and two parameters Q and k, find a CVRP tour for which the total weight of the cycles is minimum.

CVRP tour in a weighted directed or undirected complete graph with vertex set $\{0, 1, \ldots, n\}$, with a demand $d_i \geq 0$ for $i = 1, 2, \ldots, n$, and two parameters Q and k: a collection of k cycles $C_1, C_2, \ldots, C_k$, which contain all the vertices, pairwise intersect only in vertex 0, and satisfy $\sum_{i \in V(C_j)} d_i \leq Q$ for each $j = 1, 2, \ldots, k$.

distance matrix of an instance of STSP (ATSP): the matrix $D = [d_{ij}]$, where d_{ij} is the weight of the edge between vertices i and j (arc from i to j).

Euclidean TSP: the special case of STSP in which the vertices are points in the Euclidean plane and the weight on each edge is the Euclidean distance between its endpoints.

exact algorithm: an algorithm that solves a certain optimization problem to optimality.

Generalized TSP: Given a weighted complete digraph $\overleftrightarrow{K}_n$ and a partition $V_1, \ldots, V_k$ of its vertices, find a minimum-weight cycle containing exactly one (at least one) vertex from each set V_i, $i = 1, \ldots, k$.

Symmetric TSP (STSP): Given a complete (undirected) graph K_n with weights on the edges, find a minimum-weight hamiltonian cycle in K_n.

triangle inequality on a weighted digraph (or graph): the condition $d_{ij} + d_{jk} \geq d_{ik}$ for all distinct vertices i, j, k, where d_{ij} is the weight of the arc from vertex i to vertex j (of the edge between i and j).