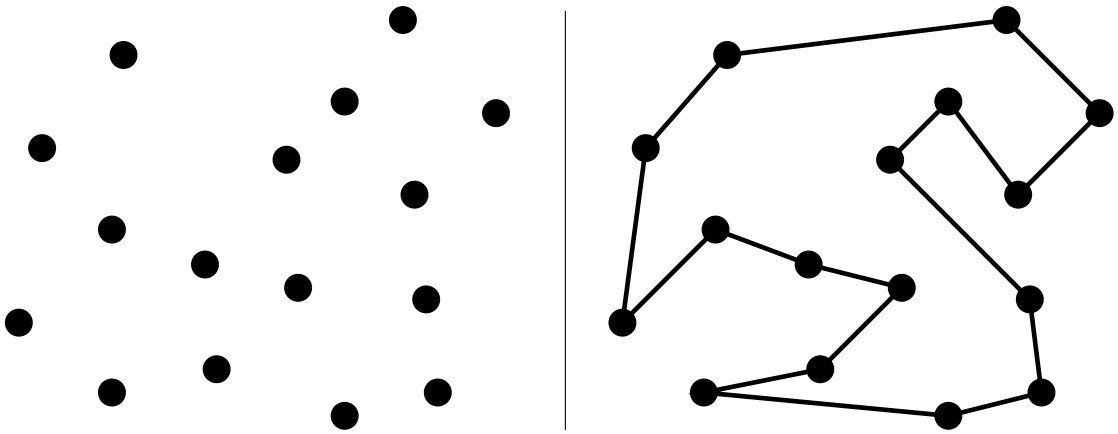

INPUT OUTPUT

16.4 Traveling Salesman Problem

Input description: A weighted graph G.

Problem description: Find the cycle of minimum cost, visiting each vertex of G exactly once.

Discussion: The traveling salesman problem is the most notorious NP-complete problem. This is a function both of its general usefulness and the ease with which it can be explained to the public at large. Imagine a traveling salesman planning a car trip to visit a set of cities. What is the shortest route that will enable him to do so and return home, thus minimizing his total driving?

The traveling salesman problem arises in many transportation and routing problems. Other important applications involve optimizing tool paths for manufacturing equipment. For example, consider a robot arm assigned to solder all the connections on a printed circuit board. The shortest tour that visits each solder point exactly once defines the most efficient route for the robot.

Several issues arise in solving TSPs:

- *Is the graph unweighted?* – If the graph is unweighted, or all the edges have one of two possible cost values, the problem reduces to finding a *Hamiltonian cycle*. See Section 16.5 (page 538) for a discussion of this problem.

- *Does your input satisfy the triangle inequality?* – Our sense of how proper distance measures behave is captured by the *triangle inequality*. This property states that $d(i, j) \leq d(i, k) + d(k, j)$ for all vertices $i, j, k \in V$. Geometric

distances all satisfy the triangle inequality because the shortest distance between two points is as the crow flies. Commercial air fares do *not* satisfy the triangle inequality, which is why it is so hard to find the cheapest airfare between two points. TSP heuristics work much better on sensible graphs that do obey the triangle inequality.

- *Are you given n points as input or a weighted graph?* – Geometric instances are often easier to work with than a graph representation. Since pair of points define a complete graph, there is never an issue of finding a feasible tour. We can save space by computing these distances on demand, thus eliminating the need to store an $n \times n$ distance matrix. Geometric instances inherently satisfy the triangle inequality, so they can exploit performance guarantees from certain heuristics. Finally, one can take advantage of geometric data structures like kd-trees to quickly identify close unvisited sites.

- *Can you visit a vertex more than once?* – The restriction that the tour not revisit any vertex is irrelevant in many applications. In air travel, the cheapest way to visit all vertices might involve repeatedly visiting an airport hub. Note that this issue does not arise when the input observes the triangle inequality.

  TSP with repeated vertices is easily solved by using any conventional TSP code on a new cost matrix D, where $D(i, j)$ is the shortest path distance from i to j. This matrix can be constructed by solving an all-pairs shortest path (see Section 15.4 (page 489)) and satisfies the triangle inequality.

- *Is your distance function symmetric?* – A distance function is *asymmetric* when there exists x, y such that $d(x, y) \neq d(y, x)$. The asymmetric traveling salesman problem (ATSP) is much harder to solve in practice than symmetric (STSP) instances. Try to avoid such pathological distance functions. Be aware that there is a reduction converting ATSP instances to symmetric instances containing twice as many vertices [GP07], that may be useful because symmetric solvers are so much better.

- *How important is it to find the optimal tour?* – Usually heuristic solutions will suffice for applications. There are two different approaches if you insist on solving your TSP to optimality, however. *Cutting plane methods* model the problem as an integer program, then solve the linear programming relaxation of it. Additional constraints designed to force integrality are added if the optimal solution is not at an integer point. *Branch-and-bound algorithms* perform a combinatorial search while maintaining careful upper and lower bounds on the cost of a tour. In the hands of professionals, problems with thousands of vertices can be solved. Maybe you can too, if you use the best solver available.

Almost any flavor of TSP is going to be NP-complete, so the right way to proceed is with heuristics. These typically come within a few percent of the optimal

solution, which is close enough for engineering work. Unfortunately, literally dozens of heuristics have been proposed for TSP, so the situation can be confusing. Empirical results in the literature are sometime contradictory. However, we recommend choosing from among the following heuristics:

- *Minimum spanning trees* – This heuristic starts by finding the minimum spanning tree (MST) of the sites, and then does a depth-first search of the resulting tree. In the course of DFS, we walk over each of the $n-1$ edges exactly twice: once going down to discover a new vertex, and once going up when we backtrack. Now define a tour by ordering the vertices by when they were discovered. If the graph obeys the triangle inequality, the resulting tour is at most twice the length of the optimal TSP tour. In practice, it is usually better, typically 15% to 20% over optimal. Furthermore, the running time is bounded by that of computing the MST, which is only $O(n \lg n)$ in the case of points in the plane (see Section 15.3 (page 484)).

- *Incremental insertion methods* – A different class of heuristics starts from a single vertex, and then inserts new points into this partial tour one at a time until the tour is complete. The version of this heuristic that seems to work best is *furthest point* insertion: of all remaining points, insert the point v into a partial tour T such that

  $$\max_{v \in V} \min_{i=1}^{|T|} (d(v, v_i) + d(v, v_{i+1}))$$

  The "min" ensures that we insert the vertex in the position that adds the smallest amount of distance to the tour, while the "max" ensures that we pick the worst such vertex first. This seems to work well because it "roughs out" a partial tour first before filling in details. Such tours are typically only 5% to 10% longer than optimal.

- *K-optimal tours* – Substantially more powerful are the Kernighan-Lin, or *k-opt*, class of heuristics. The method applies local refinements to an initially arbitrary tour in the hopes of improving it. In particular, subsets of k edges are deleted from the tour and the k remaining subchains rewired to form a different tour with hopefully a better cost. A tour is k-optimal when no subset of k edges can be deleted and rewired to reduce the cost of the tour. Two-opting a tour is a fast and effective way to improve any other heuristic. Extensive experiments suggest that 3-optimal tours are usually within a few percent of the cost of optimal tours. For $k > 3$, the computation time increases considerably faster than the solution quality. Simulated annealing provides an alternate mechanism to employ edge flips to improve heuristic tours.

Implementations: Concorde is a program for the symmetric traveling salesman problem and related network optimization problems, written in ANSI C. This

world record-setting program by Applegate, Bixby, Chvatal, and Cook [ABCC07] has obtained the optimal solutions to 106 of TSPLIB's 110 instances; the largest of which has 15,112 cities. Concorde is available for academic research use from *http://www.tsp.gatech.edu/concorde*. It is the clear choice among available TSP codes. Their *http://www.tsp.gatech.edu/* website features very interesting material on the history and applications of TSP.

Lodi and Punnen [LP07] put together an excellent survey of available software for solving TSP. Current links to all programs mentioned are maintained at *http://www.or.deis.unibo.it/research_pages/tspsoft.html*.

TSPLIB [Rei91] provides the standard collection of hard instances of TSPs that arise in practice. The best-supported version of TSPLIB is available from *http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/*, although the instances are also available from Netlib (see Section 19.1.5 (page 659)).

Tsp_solve is a C++ code by Chad Hurwitz and Robert Craig that provides both heuristic and optimal solutions. Geometric problems of size up to 100 points are manageable. It is available from *http://www.cs.sunysb.edu/∼algorith* or by e-mailing Chad Hurrwitz at churritz@cts.com. GOBLIN (*http://www.math.uni-augsburg.de/∼fremuth/goblin.html*) implements branch-and-bound algorithms for both symmetric and asymmetric TSP, as well as a variety of heuristics.

Algorithm 608 [Wes83] of the *Collected Algorithms of the ACM* is a Fortran implementation of a heuristic for the quadratic assignment problem—a more general problem that includes the traveling salesman as a special case. Algorithm 750 [CDT95] is a Fortran code for the exact solution of asymmetric TSP instances. See Section 19.1.5 (page 659) for details.

Notes: The book by Applegate, Bixby, Chvatal, and Cook [ABCC07] documents the techniques they used in their record-setting TSP solvers, as well as the theory and history behind the problem. Gutin and Punnen [GP07] now offer the best reference on all aspects and variations of the traveling salesman problem, displacing an older but much beloved book by Lawler et al. [LLKS85].

Experimental results on heuristic methods for solving large TSPs include [Ben92a, GBDS80, Rei94]. Typically, it is possible to get within a few percent of optimal with such methods.

The Christofides heuristic [Chr76] is an improvement over the minimum-spanning tree heuristic and guarantees a tour whose cost is at most 3/2 times optimal on Euclidean graphs. It runs in $O(n^3)$, where the bottleneck is the time it takes to find a minimum-weight perfect matching (see Section 15.6 (page 498)). The minimum spanning tree heuristic is due to [RSL77].

Polynomial-time approximation schemes for Euclidean TSP have been developed by Arora [Aro98] and Mitchell [Mit99], which offer $1 + \epsilon$ factor approximations in polynomial time for any $\epsilon > 0$. They are of great theoretical interest, although any practical consequences remain to be determined.

The history of progress on optimal TSP solutions is inspiring. In 1954, Dantzig, Fulkerson, and Johnson solved a symmetric TSP instance of 42 United States cities [DFJ54]. In 1980, Padberg and Hong solved an instance on 318 vertices [PH80]. Applegate et al. [ABCC07] have recently solved problems that are twenty times larger than this. Some of

this increase is due to improved hardware, but most is due to better algorithms. The rate of growth demonstrates that exact solutions to NP-complete problems can be obtained for large instances if the stakes are high enough. Fortunately or unfortunately, they seldom are.

Size is not the only criterion for hard instances. One can easily construct an enormous graph consisting of one cheap cycle, for which it would be easy to find the optimal solution. For sets of points in convex position in the plane, the minimum TSP tour is described by its convex hull (see Section 17.2 (page 568)), which can be computed in $O(n \lg n)$ time. Other easy special cases are known.

Related Problems: Hamiltonian cycle (see page 538), minimum spanning tree (see page 484), convex hull (see page 568).