importance, apart from evident transportation and logistics areas. A classic example is in printed circuit manufacturing: scheduling of a route of the drill machine to drill holes in a PCB. In robotic machining or drilling applications, the "cities" are parts to machine or holes (of different sizes) to drill, and the "cost of travel" includes time for retooling the robot (single machine job sequencing problem).[10]

- The generalized travelling salesman problem, also known as the "travelling politician problem", deals with "states" that have (one or more) "cities" and the salesman has to visit exactly one "city" from each "state". One application is encountered in ordering a solution to the cutting stock problem in order to minimize knife changes. Another is concerned with drilling in semiconductor manufacturing, see e.g., U.S. Patent 7,054,798 (https://www.google.com/patents/US7054798). Noon and Bean demonstrated that the generalized travelling salesman problem can be transformed into a standard travelling salesman problem with the same number of cities, but a modified distance matrix.
- The sequential ordering problem deals with the problem of visiting a set of cities where precedence relations between the cities exist.
- A common interview question at Google is how to route data among data processing nodes; routes vary by time to transfer the data, but nodes also differ by their computing power and storage, compounding the problem of where to send data.
- The travelling purchaser problem deals with a purchaser who is charged with purchasing a set of products. He can purchase these products in several cities, but at different prices and not all cities offer the same products. The objective is to find a route between a subset of the cities, which minimizes total cost (travel cost + purchasing cost) and which enables the purchase of all required products.

Integer linear programming formulation

TSP can be formulated as an integer linear program.[11][12][13] Label the cities with the numbers 1, ..., n and define:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$$

Good. We have n cities to visit; we want to visit each one

For i = 1, ..., n, let u_i be a dummy variable, and finally take c_{ij} to be the distance from city i to city j. Then TSP can be written as the following integer linear programming problem:

u_i are meaningful variables:

$u_i = \begin{cases} \text{the number of steps we take} \\ \text{to reach city } i \end{cases}$

(Assume: city 1 start of tour)

$$\begin{aligned}
&\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}: && \\
&0 \leq x_{ij} \leq 1 && i,j = 1, \ldots, n; \\
&u_i \in \mathbf{Z} && i = 1, \ldots, n; \\
&\sum_{i=1, i \neq j}^{n} x_{ij} = 1 && j = 1, \ldots, n; \\
&\sum_{j=1, j \neq i}^{n} x_{ij} = 1 && i = 1, \ldots, n; \\
&u_i - u_j + n x_{ij} \leq n - 1 && 2 \leq i \neq j \leq n; \\
&0 \leq u_i \leq n - 1 && 2 \leq i \leq n.
\end{aligned}$$

$x_{ij} \neq 0, 1$

① $x_{ij} = 0$ the inequality is redundant ($u_i - u_j \leq n-1$ always)

② $x_{ij} = 1$ $u_i + 1 \leq u_j$

In your progress report you would have to explain this right after you write it.

$1 \leq u_i \leq n$ is better; both are OK

The first set of equalities requires that each city be arrived at from exactly one other city, and the second set of equalities requires that from each city there is a departure to exactly one other city. The last constraints enforce that there is only a single tour covering all cities, and not two or more disjointed tours that only collectively cover all cities. To prove this, it is shown below (1) that every feasible solution contains only one closed sequence of cities, and (2) that for every single tour covering all cities, there are values for the dummy variables u_i that satisfy the constraints.

To prove that every feasible solution contains only one closed sequence of cities, it suffices to show that every subtour in a feasible solution passes through city 1 (noting that the equalities ensure there can only be one such tour). For if we sum all the inequalities corresponding to $x_{ij} = 1$ for any subtour of k steps not passing through city 1, we obtain:

$$nk \leq (n-1)k,$$

which is a contradiction.

Software note: all the $i \neq j$ above are clumsy to program. In Gurobi it is simpler to enforce $x_{ii} = 0$.

It now must be shown that for every single tour covering all cities, there are values for the dummy variables u_i that satisfy the constraints.

Without loss of generality, define the tour as originating (and ending) at city 1. Choose $u_i = t$ if city i is visited in step t (i, t = 1, 2, ..., n). Then

$$u_i - u_j \leq n - 1,$$

since u_i can be no greater than n and u_j can be no less than 1; hence the constraints are satisfied whenever $x_{ij} = 0$. For $x_{ij} = 1$, we have:

$$u_i - u_j + nx_{ij} = (t) - (t+1) + n = n - 1,$$

satisfying the constraint.

Computing a solution

The traditional lines of attack for the NP-hard problems are the following:

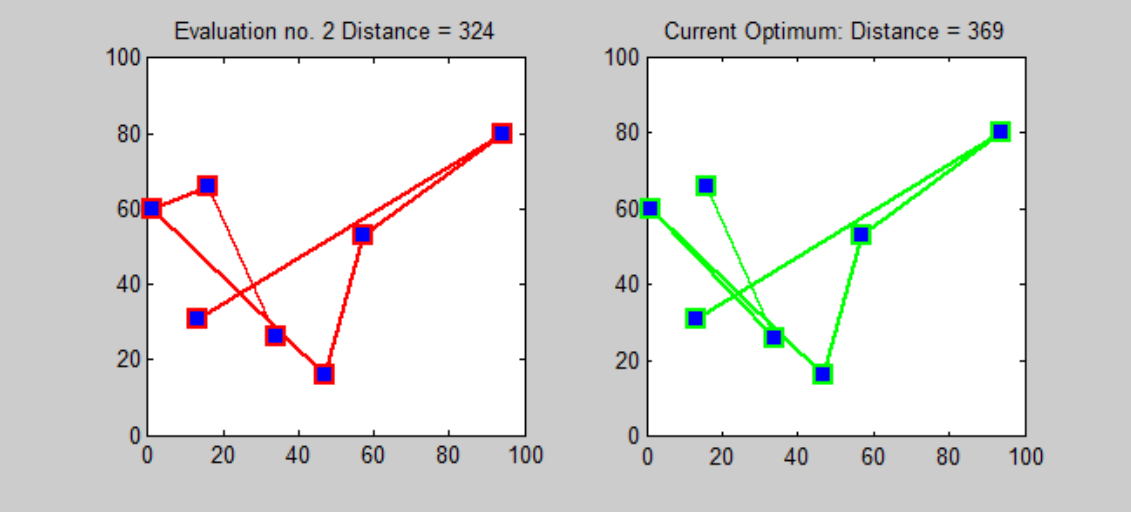
- Devising exact algorithms, which work reasonably fast only for small problem sizes.
- Devising "suboptimal" or heuristic algorithms, i.e., algorithms that deliver either seemingly or probably good solutions, but which could not be proved to be optimal.
- Finding special cases for the problem ("subproblems") for which either better or exact heuristics are possible.

Exact algorithms

The most direct solution would be to try all permutations (ordered combinations) and see which one is cheapest (using brute force search). The running time for this approach lies within a polynomial factor of $O(n!)$, the factorial of the number of cities, so this solution becomes impractical even for only 20 cities.

One of the earliest applications of dynamic programming is the Held–Karp algorithm that solves the problem in time $O(n^2 2^n)$.[14]

Improving these time bounds seems to be difficult. For example, it has not been determined whether an exact algorithm for TSP that runs in time $O(1.9999^n)$ exists.[15]



Solution to a symmetric TSP with 7 cities using brute force search. Note: Number of permutations: (7-1)!/2 = 360

Other approaches include:

- Various branch-and-bound algorithms, which can be used to process TSPs containing 40–60 cities.
- Progressive improvement algorithms which use techniques reminiscent of linear programming. Works well for up to 200 cities.
- Implementations of branch-and-bound and problem-specific cut generation (branch-and-cut[16]); this is the method of choice for solving large instances. This approach holds the current record, solving an instance with 85,900 cities, see Applegate et al. (2006).

An exact solution for 15,112 German towns from TSPLIB was found in 2001 using the cutting-plane method proposed by George Dantzig, Ray Fulkerson, and Selmer M. Johnson in 1954, based on linear programming. The computations were performed on a network of 110 processors located at Rice University and Princeton University (see the Princeton external link). The total computation time was equivalent to 22.6 years on a single 500 MHz Alpha processor. In May 2004, the travelling salesman problem of visiting all 24,978 towns in Sweden was solved: a tour of length approximately 72,500 kilometres was found and it was proven that no shorter tour exists.[17] In March 2005, the travelling salesman problem of visiting all 33,810 points in a circuit board was solved using *Concorde TSP Solver*: a tour of length 66,048,945 units was found and it was proven that no shorter tour exists. The computation took approximately 15.7 CPU-years (Cook et al. 2006). In April 2006 an instance with 85,900 points was solved using *Concorde TSP Solver*, taking over 136 CPU-years, see Applegate et al. (2006).