Parabola Volume 37, Issue 2 (2001)

The Travelling Salesman Problem and Computational complexity

Rob Womersley[1]

If a salesman, starting from his home city, is to visit exactly once each city on a given list and then return home, it is plausible for him to select the order in which he visits the cities so that the total of the distances travelled in his tour is as small as possible. [4]

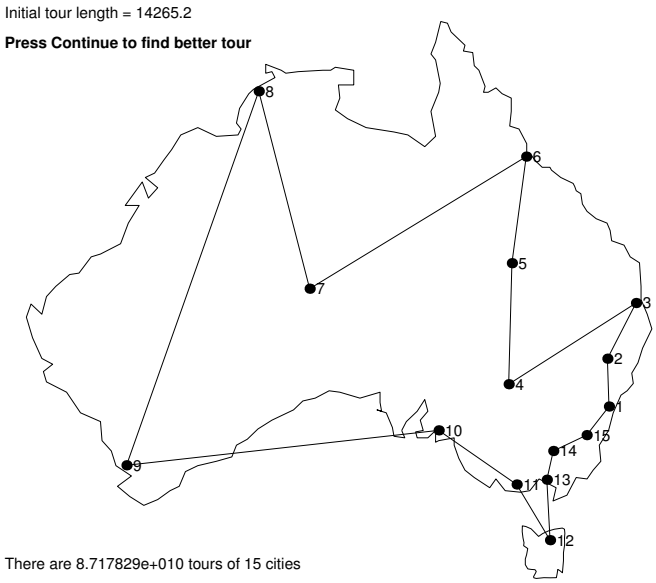


Figure 1: A tour of 15 cities in Australia

The *Travelling Salesman Problem* (TSP) is to find a tour of n cities which minimizes the total cost of the tour. A *tour* must visit all n cities exactly once, and return to the starting city. A tour of some cities in Australia is illustrated in Figure 1. The "cost" associated with travelling between cities may be the distance travelled or the actual cost of transport (air or bus fares or hiring a car plus petrol), or some other criteria. The cases $n = 1, 2$ don't allow for any choice, so we will think of n being at least 3. In fact we are interested in solving problems for as large a value of n as possible!

The TSP became widely know in the mathematical literature in the 1950's, but similar problems date back to the 18th century [4]. The TSP and closely related problems

[1]Rob Womersley is a Senior Lecturer in Applied Mathematics at UNSW.

arise in a wide range of applications, including vehicle routing, computer wiring, cutting wallpaper, job sequencing and clustering data. The TSP has even been used to try to maximize the fun at an amusement park [1]. Even more importantly it is an example of a class of problems which are easy to state but very hard to solve in practice. It is an example of an NP complete problem, which were discussed by Terry Tao in his Parabola article [7].

Label the cities $1, \ldots, n$ and let c_{ij} represent the "cost" of travelling from city i to city j. This "cost" could just be the distance between cities, in which case $c_{ij} = c_{ji}$, and the TSP is said to be *symmetric*. In a non-symmetric TSP the cost of travelling from city i to city j is not necessarily the same as travelling from city j to city i. An example would be an upriver journey by boat from city i to city j which is slower than the downriver journey from city j to city i.

There are many variations on the same basic theme: maybe you can only reach some of the cities from a given city (there is no direct Sydney to London flight yet - you have to go via Singapore or Bangkok or some other city) or the geometry may be different (when travelling around the world, cities are on the surface of a sphere).

Brute Force and Ignorance?

First let us count the number of possible tours of n cities. We start at a particular city – it does not matter which one. At that point we have a choice of $n-1$ cities to go to, as we are not allowed to visit a city more than once. After reaching the second city we have a choice of $n-2$ cities to go to. Similarly after visiting 3 cities we have a choice of $n-3$ cities to go to. Finally after visiting all n cities we complete the tour by returning to the city we started at. Thus for n cities the total number of tours is $(n-1)!$.

Another way of thinking of a tour is as a permutation v of the integers $\{1, 2, \ldots, n\}$, so that journey i is from city $v(i)$ to city $v(i+1)$. For example $\{3, 1, 5, 2, 4\}$ is a permutation of $\{1, 2, 3, 4, 5\}$. To complete a tour we need to add the condition that $v(n+1) = v(1)$. If you have a symmetric TSP then there are fewer different tours (how many?).

To evaluate the cost of a tour we simply have to add up the costs of the n journeys between the cities, giving a total cost of

$$\sum_{i=1}^{n} c_{v(i)v(i+1)} = c_{v(1)v(2)} + c_{v(2)v(3)} + \cdots + c_{v(n-1)v(n)} + c_{v(n)v(1)}.$$

Given a tour v, the time to calculate the cost of the tour is proportional to n.

You might think of simply enumerating the finite set of possible tours, calculating their cost, and picking the one with lowest cost. This would take n operations for each of the $(n-1)!$ tours, for a total of $n!$ operations. You can do this only for very small numbers of cities!

What is a large travelling salesman problem? Currently the fastest computer in the world can do around 1 teraflop or 10^{12} operations per second (a more typical 1 GHz PC at best could do 1 gigaflop or 10^9 operations per second) . Using simple enumeration to solve a TSP on this computer estimate

- how long it will take to solve a TSP with $n = 50$ cities?
- the largest TSP that can be solved in one hour?

The time taken to solve a TSP by simple enumeration is proportional to $n!$. As this computer can do 10^{12} operations per second, a 50 city TSP will take approximately

$$\frac{50!}{10^{12}} \approx \frac{3 \times 10^{64}}{10^{12}} = 3 \times 10^{52} \text{ seconds.}$$

As physicists estimate the life of the universe to be around 10 thousand million years or around 10^{17} seconds, solving a 50 city TSP by simple enumeration is clearly impractical.

In one hour this computer can do around 3.6×10^{15} operations. Some simple experimentation with a calculator shows that the largest TSP that can be solved by enumeration in this time has only 17 cities!

Yet TSPs with hundreds of cities have been solved! Clearly they used much smarter algorithms than simple enumeration.

Linear and integer programming

Define the n^2 variables

$$x_{ij} = \begin{cases} 1 & \text{if you travel from city } i \text{ to city } j, \\ 0 & \text{otherwise.} \end{cases}$$

Such variables are known as zero-one variables, as they can only take the value 0 or the value 1. The total cost of moving as described by the x_{ij} variables is then

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} = c_{11}x_{11} + c_{12}x_{12} + \ldots c_{1n}x_{1n} + \ldots + c_{nn}x_{nn}.$$

The difficulty is adding constraints or restrictions to the n^2 variables to ensure that they represent a tour of the n cities. As we cannot travel from city i to city i we must have $x_{ii} = 0$ for all $i = 1, \ldots, n$. As you must travel from city i to exactly one other city

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \text{for all } i = 1, \ldots, n. \tag{1}$$

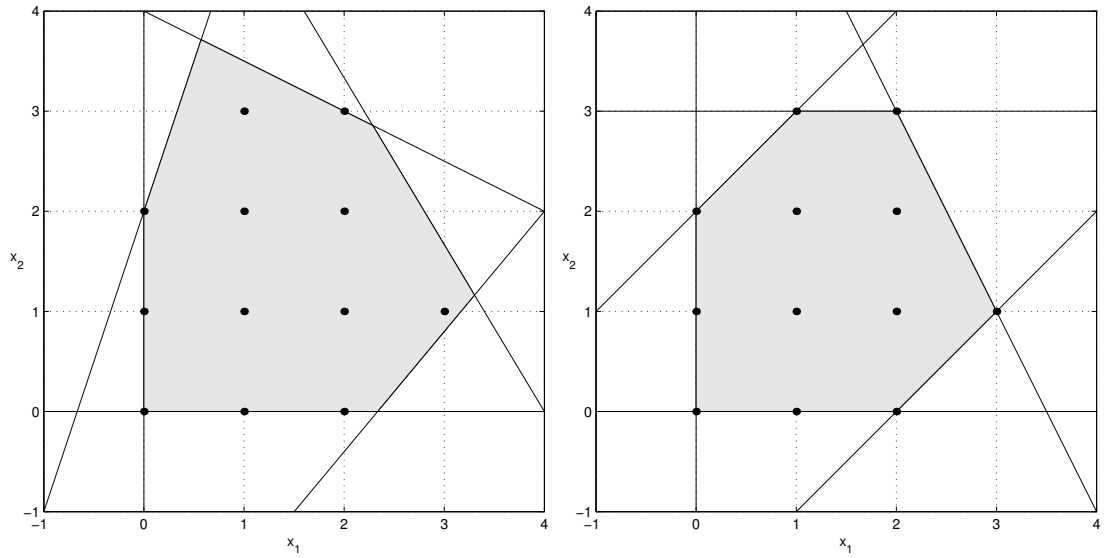
Also as you must reach city j from exactly one other city

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \text{for all } j = 1, \ldots, n. \tag{2}$$

These constraints characterize *assignment problems* which can be solved efficiently, that is in time which is a polynomial of the size n of the problem. However they are not enough for the TSP as they still allow subtours (for example two subtours of the cities

in Figure 1 would be $\{1, 2, 3, 15, 9, 10, 11, 12, 13, 14, 1\}$ and $\{4, 5, 6, 7, 8, 4\}$). Subtours can be eliminated by adding an extra 2^n constraints. For large n this is a lot of extra constraints. There is still active research in finding the best set of constraints which characterize a tour to add to the problem.

If you allow the variables x_{ij} to be non-negative, $x_{ij} \geq 0$, rather than just 0 or 1, then you get a *linear programming relaxation* of the TSP. Linear programming, which minimizes (or maximizes) a linear function of the variables, subject to linear equality and inequality constraints on the variables, is widely used in economics, management and engineering. Since the work of Karmarker in 1984 and the development of interior point methods, linear programs can be solved in polynomial time. Linear programming relaxations of the TSP provide a lower bound on the minimum cost tour. The constraints (1) or (2) together with $x_{ij} \geq 0$ for all i, and j imply that $x_{ij} \leq 1$ for all i and j. Unfortunately you may not get a solution to the TSP as the optimal point produced by the linear program may have fractional values for x_{ij}.



(a) Feasible region and integer points (b) Convex hull of the integer points

Consider the *feasible region* of the x_1–x_2 plane satisfying the inequalities $x_1 \geq 0$, $x_2 \geq 0$, $x_1 + 2x_2 \leq 8$, $6x_1 - 5x_2 \leq 14$, $5x_1 + 3x_2 \leq 20$ and $-3x_1 + x_2 \leq 2$. This is the shaded region in Figure 2a. We use the notation x_1 and x_2, rather than x and y, as we want to be able to generalize to hundreds or thousands of variables $x_1, x_2, x_3, x_4, \ldots$. Linear programming minimizes a linear function of the variables, such as $-7x_1 - 6x_2$, over all points satisfying the linear inequalities (the shaded region).

Integer programming solves optimization problems where the variables can only take integer values. The feasible integer points are indicated by the dots in Figures 2a and 2b. The solution of the linear programming problem is always attained at a vertex of the feasible region. However the vertex may not be integer. To solve the integer programming problems we would like to have an automatic procedure for generating the inequalities $x_1 \geq 0$, $x_2 \geq 0$,$x_2 \leq 3$, $-x_1 + x_2 \leq 2$, $2x_1 + x_2 \leq 7$ and $x_1 - x_2 \leq 2$, illustrated in Figure 2b. This region is the *convex hull* of the integer feasible points. It is the

intersection of all the linear inequalities that contain all the the integer points. A linear program with these inequalities has a solution in which each component is an integer. However it is difficult to automatically generate the inequalities describing this convex hull, especially when working with many variables.

Smart heuristics

We need to think what we mean by *solve* a TSP. Mathematically we mean finding a tour v^* for which we can guarantee there is no tour of lower cost. In practice we may be quite happy to have a "good" tour – say one that has a cost within a few percent of the optimal tour. You may even be satisfied with a tour which has a high probability of being the optimal tour. Good solutions, but ones which you cannot guarantee they are optimal, have been found for much larger problems.

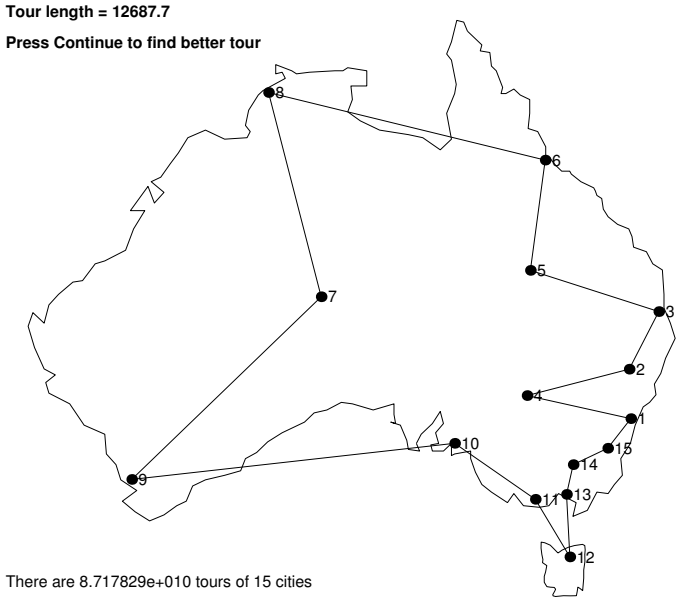


Figure 2: A better tour of 15 cities in Australia

Heuristics are simple rules used to search for better tours or eliminate bad tours. Looking at the tour in Figure 1 suggests that it is not a good tour. A better tour is illustrated in Figure 2, but this is still not the best tour. Heuristics can produce good tours, but they can't prove that a tour is optimal.

Quantum Computing

The complexity ideas discussed here are based on classical computers. Quantum computers, which are still more science fiction than reality, are being actively investigated by researchers around the world (see [6, 5] for example). They have the potential to make some tasks that are currently impossible on even the fastest computers a real possibility.

References

[1] R. Bosch, M. Cardiff and G Hughes, Maximizing fun at an amusement park, The UMAP Journal, 21,4 (2000) 483-498.

[2] M. Garey and D. Johnson, Computers and Intractability, A Guide to the Theory of NP-completeness, (W. H. Freeman and Co., 1979).

[3] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica, 4 (1984) 373-395.

[4] M. Lawler et al eds., The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization, (Wiley, 1985).

[5] Centre for Quantum Computation, `http://www.qbit.org/`[2].

[6] Semiconductor Nanofrabrication Facility, University of New South Wales, `http://www.snf.unsw.edu.au/`[3].

[7] T. Tao, Does P = NP? A real-life mathematical problem, Parabola 36,3 (2000) 2-7.

[2] Editorial note, February 2014: this is now a dead link.

[3] Editoral note, February 2014: this is now a dead link. But see also http://www.cqc2t.org/facilities/unsw/snf.