TSP

Perrin Westrich

Executive Summary

In the TSP a salesman has a list of cities to travel to and must visit each city exactly once using the shortest overall path. The problem is easy to conceptualize, because the difficulty does not lie in the conceptualization, but in the computational complexity as the number of cities grows.

One approach to solving the problem would be to try every possible path. But again, the computational complexity gets in the way. As the number of cities grows so does number of paths. There are a few ways to reduce the number of paths to choose; however, it takes computation time to reduce the number of permutations. What I am going to try is to compare a brute force method that looks at every permutation, with a method that throws out half of the permutations, before they are calculated. I compare the time it takes to do the exact same problem on both the brute force and the modified brute force search.

The result of this experiment shows that the modified brute force search is slightly faster, but not significantly faster than the brute force search. The modified brute force search took about 78% of the amount of time that it took the brute force search to compute a tsp problem. The 78% is not much of an improvement, but it is still an interesting perspective on computing the tsp.

Problem Description

The traveling salesmen problem(TSP) is an optimization problem that has simple premises, but is very computationally complex.<Modern heuristics> "The problem is NP hard: There are no known algorithms for finding perfect solutions that have a complexity that grows as only a polynomial function of the number of cities."<Modern heuristics> In the TSP, a salesman has a list of cities to travel to and must visit each city exactly once by taking the shortest overall path. The most straight forward approach would be to try all the possible paths. That would give a search space of the set of permutations of n cities, since any permutation would be visiting each city once.<Modern heuristics> The main difficulty in trying to solve this problem is that the computational complexity increases incredibly fast. With N number of cities there are N! number of paths that the salesmen can take. Unfortunately the straight forward approach becomes increasingly infeasible.<Traveling>

Analysis Technique

A brute force approach has a search space of every permutation of the cities. This is n! number of possible paths where n is the number of cities. This can be reduced to (n-1)! since you have to make a complete circuit. This can be done because it does not matter if the starting city is given or not. Since it is a circuit, any one of the cities in the path can be the starting city, and that particular path would still have the same distance. Thus reducing it to (n-1)!. A brute force search would then take all the possible (n-1)! Of the combinations and add up each ones total length, and keep track of the shortest path.

One possible way to find a pattern is to find the permutations of a set by hand and listing them in alphabetical order. In figure 1, the first group has the first letter fixed and the other three letters change. Inside that group, there are multiple groups where the second letter is fixed and last two letters are changed. These different layers I will call factorial groupings, because when three letters change there are 3! number of different possibilities therefore, I call these the 3! groupings.

By indexing all the permutations starting from 0 going up to n!-1, and splitting the permutations into the factorial groupings, it is possible to show that starting from the highest level factorial grouping and continuing inward you can represent the index as a list of factorial groupings numbers. This is a nonstandard base of numbers since each grouping layer does not increase by a fixed power like it does in base 10. Each digit has a place for example the 1! place the 2! place and so on. This base factorial is sometimes referred to as the factoradic numbering system. “The factoradic numbering system is unambiguous. No number can be represented in more than one way because the sum of consecutive factorials multiplied by their index is always the next factorial minus one:”<Factoradic> “There is a natural mapping between the integers 0, ..., *n*! - 1 (or equivalently the factoradic numbers with *n* digits) and permutations of *n* elements in lexicographical order, when the integers are expressed in factoradic form. This mapping has been termed the Lehmer code or Lucas-Lehmer code (inversion table).”<Factoradic>

Conversion to this base is the same process as converting to any other base. Sometimes you will see factoradic least significant digit always be 0, or 0*0!. I choose to leave this off since it is always the same thing, and can be implied.

Since this is a symmetric tsp problem that we are dealing with, a brute force search can be improved by removing all the permutations that are reflections. This is done in figure 1(The * means it is a reflection of something higher on the list), by looking at the last letter and seeing if it is a letter that was first, earlier in the list. For a person to do this it is relatively straight forward. But on a machine, the machine has to parse out the last letter of the string and compare it to all the other letters that previously came first. The benefit of not having to add up the lengths of the permutations that were cut out, is quickly outweighed by the computational cost of finding out if the paths are a reflection of a previous path.

It would be nice if I could find out if a permutation is a reflection of another path before it is calculated. Looking at figure 1, it can be seen that A will always work its way to the end at the same position of a 3! group after the group in which A leads. This will happen to any letter because the list is in alphabetical order, and after a letter has been in the front for a whole group it will always be in the same starting position at the beginning of the 3! group. The same thing would happen if you had a larger number of cities.

Figure 2 depicts a three city list of permutation with the reflections marked. Looking at the 2! groups, there are three different patterns labeled 0-2. Looking back at figure 1 which has the four city problem, there are the three reflection patterns from the three city problem repeated in a noticeable way. The first 3! group has the 0 pattern repeated three times. The second 3! group has the 0 pattern once, and the 1 pattern twice. It continues this way; each pattern being repeated four times. One way to look at it is, you have three patterns, and each consists of 2! , and each of these patterns is repeated four times(4*3*2!=4! which is how many different permutations there are.) Figure 3 shows this patterned relationship for the four city problem example. Each column in the table represents a factorial group, and the specific number shows which of the sub pattern is used. Figure 4 is the same type of table, but extended for an eight city problem. Notice how the table in figure 3 fits inside the table in figure 4. Because of that relationship, a looping structure can be set up so that it reduces the placement until your down to just two patterns, a 0 or a 1. Once it is reduced to a 0 or a 1, a choice can be made on whether to compute all the 0 or the 1 permutations. Reducing it to smaller and smaller tables is directly related to the factoradic representation of the numbered permutations.

Using this modified brute force approach is only beneficial if the amount of time it takes to find out if something is a reflection, is faster than the time it takes to compute a given permutation, and to add up its length. To test this, I will set up two programs, one of which uses a brute force search and the other uses a modified brute force search. They will both compute tsp problems with the same input. The input will be the number of cities, and a nxn matrix where the intersection of a row and a column is the distance between the given row and column city, and n is the number of cities.

Assumptions

Symmetric TSP problem…

Results

The first attempt at comparing the two programs resulted in the modified brute force search taking about 50 seconds to compute an 11 city problem. The pure brute force search took about 100 seconds. This shows a noticeable difference of the speed of the modified brute force search by a factor of two in comparison to the pure brute force search.

After looking at the output files and comparing the answers I realized that there was a lot of unnecessary output in the file. The problem states “Find the shortest path.” It does not matter if there are multiple with the same length path only one of the shortest paths is needed. I was initially printing out the permutation and the path length for all of the paths that the programs computed, which the file size was about 37 MB for the brute force search, and about half that for the modified brute force search. After discovering those initial problems I went back and reworked the programs to only print out to the file the length of the shortest path.

After modifying both programs to only output the fastest route, both programs speed improved considerably. The brute force search version took approximately 5 seconds for the 11 city problem, while the modified brute force search took about 4 seconds for the same problem. Further testing was conducted to make sure that these times where accurate. I then set up a twelve city problem and predicted the amount of time they would take, I predicted that the brute force search would take 60 seconds, and the modified brute force search would take 48 seconds. The result of the actual testing the brute force search was exactly 60 seconds, and the modified brute force search was very close to the estimate with 47 seconds.

The current incarnation of the modified brute force takes about .784 times the amount of time as the pure brute force search. Not an incredible amount of improvement by any means, but this method does offer an interesting perspective on the tsp problem.

Issues

The modified brute force search algorithm relies on the principle that the tsp is a symmetric tsp problem.

Appendices

Traveling Salesmen Problem. Retrieved November 12, 2006, from
http://en.wikipedia.org/wiki/Travelling_salesman_problem

Factoradic. Retrieved November 18, 2006, from
http://en.wikipedia.org/wiki/Factoradic

Zbigniew Michalewicz, and David B. Fogel. How to Solve It: Modern Heuristics. Springer-Verlag Berlin Heidelberg 2000

Figure 1
ABCD

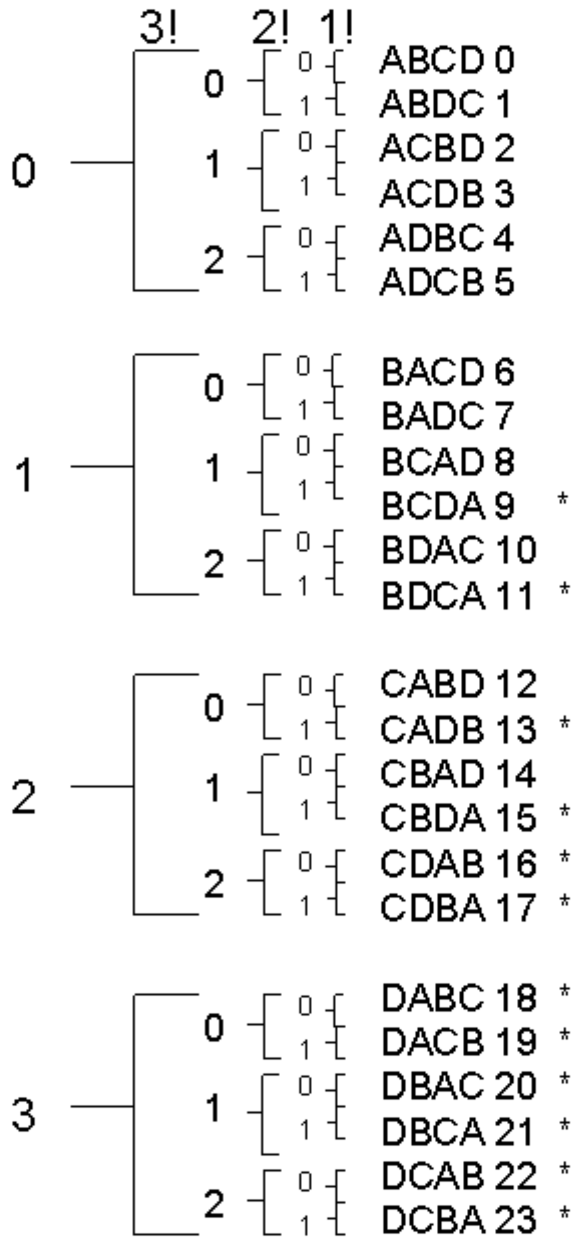

Figure 2
ABC

0
ABC
ACB

1
BAC
BCA *-

2
CAB *--
CBA *-

Figure 3
Reflection table for the 4 city problem example

0	0	1	2
0	1	1	2
0	1	2	2

Figure 4
Reflection table for an 8 city problem

0	0	1	2	3	4	5	6	7
0	1	1	2	3	4	5	6	7
0	1	2	2	3	4	5	6	7
0	1	2	3	3	4	5	6	7
0	1	2	3	4	4	5	6	7
0	1	2	3	4	5	5	6	7
0	1	2	3	4	5	6	6	7
0	1	2	3	4	5	6	7	7