HW 12. Longest Path Problem

EE3980 Algorithms

104061212 馮立俞

June 2, 2018

Introduction

Given N cities and the distances between different cities stored with an $N * N$ adjacency matrix, we are required to find the longest path to travel each city only once. Such path needs to be a Hamiltonian cycle. The problem can be solved using brute force approach. Yet its non-polynomial complexity is somewhat frightening.

Approach

Brute Force

We can use brute force approach to exhaust all feasible paths for the problem. To generate all permutation of N elements, Algorithm from Narayana Pandita can be utilized. The procedures are:

---

1. Find the largest index j such that $A[j] < A[j+1]$. If no such index exists, the permutation is the last permutation.
2. Find the largest index k such that $A[j] < A[k]$.
3. Swap $A[j]$ with $A[k]$.
4. Reverse the sequence from $A[j+1]$ up to and including the last element $A[N-1]$.

Or, it can also be realized following below pseudo code.

```
Algorithm Pandita(n, Perm[0..n-1]){
    int k, j, s, t;

    Sort(n, Perm[0..n-1]) // Sort to nondecresing order
    while (true){
        print Perm        // a new permutation found
        Compute_Cost();
        //Bound(); Skip();

        k := n-1
        while(k > 0 and Perm[k-1] >= Perm[k])  k := k-1
        if (k = 0) then return Done
        j := n-1
        while (Perm[k-1] >= Perm[j]) j := j-1      // j can't go below k
        Swap(Perm[k-1], Perm[j])
        s := k; t := n-1     // reverse Perm[k..n-1]
        while (s < t){
            Swap(Perm[s], Perm[t])
            s := s+1; t := t-1
        }
    }
}
```

We can find out the path with maximum cost from the generated $N!$ permutations (Actually $(N-1)!$, since the salesperson always start his trip from the first city on the list). Yet if we execute this algorithm on work station, it will only terminate in reasonable time when $N = 5$ or $N = 10$. For $N = 15$, a conservative estimation would take six days to finish.

Reduce Operation Using Bound

However, with proper calculation of upper bound for the cost, we can save some effort. If the bound is lower than current longest path, we'll simply skip and move on to the next permutation / path.

Calculating Upper Bound

When a new permutation is generated by Pandita's algorithm mentioned above, besides checking the total cost of the path, we also split the permutation into two parts to calculate the bound in order to skip unnecessary trials. In the first part, we calculate the cost to travel the path indicated by first part. Then, in the second part, we sum up the maximum cost to leave a city. That is, for a city i in second part, we find the largest element in row i in the adjacency matrix, the path indicated by the element shouldn't lead to cities in the first part, though. If the sum of results from first and second part is less than current maximal path, we'll skip all the permutations composed by cities in second part. Since no matter how we arrange them, the total path would not exceed current maximum.

Skip permutations in Pandita's algorithm

If we observe the output of Pandita's algorithm, we can see the first permutation is $1, 2, ..., N-1, N$, and the last permutation is $N, N-1, ..., 2, 1$. An easy way to jump from the first permutation to the last is to sort the numbers in decreasing order. The discipline to skip the whole permutation applies to skipping the latter part of the permutation as well. Therefore, if we want to skip the permutations for the above-mentioned second part, we could simply sort them in decreasing order. Skipping is pivotal for speeding up the process.

Results and Analysis

The execution time is tabulated as below. Cases in which N are above 15 take too much time to wait.

Table 1: CPU time taken for two approaches. (in seconds)

N	$BruteForce$	$Bound$
5	0.000	0.000
10	0.77	0.052
15	x	48.581

Because solving this problem takes non-polynomial time, and verifying a longest path means actually solving the problem again, which takes non-polynomial time. This is therefore a NP-hard problem. Such long time of execution is not surprising.

However, for $N = 10$ it is shown that the implementation of bound has introduced about 14x of speed up. The improvement is more when N is larger.

Observations and Conclusion

1. NP-hard problems' execution time grow exponentially with task size.
2. The algorithm's performance may be optimized more by devising a tighter upper bound.