Group Number ___12_________

CS 325

Summary of TSP Results

Test Case	**Minimum Tour length**	**Time**
1	5333	real 0m0.298s user 0m0.289s sys 0m0.009s
2	7921	real 0m1.509s user 0m1.502s sys 0m0.004s
3	13157	real 0m12.759s user 0m12.721s sys 0m0.016s
4	18334	real 1m2.065s user 1m1.936s sys 0m0.028s
5	25550	real 2m59.039s user 2m58.740s sys 0m0.028s
6	36750	real 2m59.074s user 2m58.795s sys 0m0.008s
7	58024	real 2m59.146s user 2m58.736s sys 0m0.124s

Brandon Swanson – Group 12
12/4/15

CS 325 Project 4 TSP approximation

Description of Algorithm

My algorithm begins by constructing a tour of all the cities using the Nearest Neighbor Heuristic and then improves on that tour using the 2-opt technique of removing two edges and replacing them if the replaced edges total distance is less than that of the two edges removed. In this process there is only one way of reconstructing the tour after removing the edges and in a data structure of sequentially stored vertexes on the tour it amounts to reversing a subsection of vertexes. I have made several attempts to improve the speed and effectiveness of this scheme that I will elaborate on here.

The algorithm begins by gathering the nearest M neighbors of each vertex. This is a O(n^2 M) operation but it only needs to be performed once and greatly improves the speed of generating tours using the nearest neighbor heuristic and prunes the search space for 2-opt edge swapping, giving a nice improvement to speed with a small trade off on the quality of the approximation. Some testing was done on what an optimal value of M was for the competition set and 10 neighbors was chosen as the best.

After each vertex has a set of its M nearest neighbors this information is used to generate a tour using the Nearest Neighbor Heuristic. That is the algorithm starts with any vertex in the set and then adds the nearest vertex to the last added vertex that is not already in the tour. This is greatly aided by the already generated list of nearest neighbors, in the worst case the most recently added vertex's set of nearest neighbors has already been added to the tour in which case it must search the remaining vertexes, but in the average case it is an O(M) operation to add the next vertex to the tour with M being a relatively small value between 5-20.

Now that a tour has been generated it is improved upon by making 2-opt swaps, beginning with the first city in the tour potential edges are examined by searching for cases in which exchanging one of the connected vertex's with one from the set of closest neighbors. Each vertex's set of neighbors is considered and then this entire process is iterated over until no more improvements can be found.

The tours are maintained as a list of city objects and each city's nearest neighbors list is a list of object pointers, for quickly creating the tour and performing 2-opt swaps each city has its own index field that is updated on each move or reversal. This makes creating a tour or finding the position in the tour of a vertex/edge to be swapped much faster (although with a cost of maintaining this index).

The first version of the algorithm chose the root vertex for the tour at random, which produced varying length original tours and 2-opt optimized length tours. But some testing revealed that the original length of the heuristic generated tour had little bearing on the length of the final optimized tour length. In other words the shortest Nearest Neighbor tour did not produce the best tour after 2-opt optimization.

With this observation in mind and the fact that the one time expensive calculation of M nearest neighbors for each vertex allowed for quickly generating new tours the algorithm was changed to generate a tour starting from each possible vertex and then 2-opt optimize each of these tours, outputting only the best discovered tour.

Finally the algorithm was modified for the purposes of the competition to be a psuedo Approximation Scheme. A time limit can be supplied in seconds as the second command line parameter and the program will terminate when it has exceeded that time limit, halting on either a 2-opt iteration or before generating a new tour. It will then output the best tour it has found in the time it was allowed to run. However there is not a linear relationship to the quality of the approximation and the time it runs only a likelihood of improvement. It is possible that the first tour it generates and optimizes will be the best one it finds.

Pseudo Code
(on following page)

```
//assume defined as presumed
//distance(u,v)  or dist(u,v)
//tourdistance(tour)

//cities = [cityobjects{x,y,neighbors}]


//takes unorderd cities
//modifies cities updating their neighbors list to nearest M
NEAREST_NEIGHBORS(cities):
    for all pairs (u,v) in cities:
        if |u.neighbors| < NUM_NEIGHBORS: u.neighbors.add(v)
        else if dist(u,v) < max(dist(u,u.neigbor[i]) for all i):
            replace greatest distance u.neighbor with v


//takes unorderd cities
//modifies order to a tour order
NEAR_NEIGHBOR_TOUR(cities, rootindex):

    swap cities[rootindex],cities[0]

    for u = 0 to |cities|-1:
        if index of all cities[u].neighbors < u:
            vindex = indexof nearest vertex in cities[u+1:]
        else:
            vindex = indexof nerest cities[u].neighbor not yet in tour //index > u

        swap cities[u+1], cities[vindex]


//takes a list of cities in a tour order
TWO_OPT(tour):

    for a in tour:
        for c in a.neighbors:
            b = vertex adjacent to a and between a and c
            d = vertex adjacent to c and not between a and c

            //removing edges a,b and c,d will result in creating edges a,c and b,d
            //                                        and reversing vertexes b---c
            //only do so if the distance of the edges removed is greater then the ones created

            if b is not c and dist(a,b) + dist(c,d) > dist(a,c) + dist(b,d):
                reverse tour[b-c]


TSP_APPROX_TOUR(cities):
    bestTour = INFINITY,{}

    NEAREST_NEIGHBORS(cities)


    for i = 1 to |cities|:

        NEAR_NEIGHBOR_TOUR(cities, cities[i]):

        while tourdistance(cities) is decreasing:
            TWO_OPT(cities)


        if tourdistance(cities) < bestTour:
            bestTour = tourdistance(cities), {record of cities order}


    return best_tour
```

Best Tours for Provided Examples

Example Case	Minimum Tour length	Time
1	112727	real 0m0.788s user 0m0.778s sys 0m0.008s
2	2709	real 0m13.834s user 0m13.805s sys 0m0.008s
3	1749725	Hours*

*****unfortunately the exact time elapsed to find the best tour for example 3 is unknown, I left it running on Flip overnight with it writing to file the best tour it had found so far after each iteration of nearest neighbor tour generation and 2-opt improvements and being measured by the system time command. My plan was to let it run as long as possible before submitting the report but it seems the server was reset as the screen session it was running exists no longer. Fortunately I have a file record of the best tour it had found, just not a recording of the total amount of time it was running.

Best tours for Competition Test Cases
see attached form

Sources Consulted

- Heuristics for the Traveling Salesman Problemm, Christian Nilsson, Linkoping University https://web.tuke.sk/fei-cit/butka/hop/htsp.pdf
- University of Waterloo Concorde TSP Solver http://www.math.uwaterloo.ca/tsp/concorde/