CS5500 HW10

Chase Mortensen A01535275

November 2019

1 Description

Solve the traveling salesperson problem. The traveling salesperson problem is to visit a number of cities in the minimum distance. The following pairs of integers represent geographic locations of cities on a Cartesian grid. The cost to travel between any two cities is equivalent to the Euclidian distance between them. Your goal is to find a low-cost in a small amount of time. One of the elements of your report is a graph showing your best solution so far versus the time your algorithm spends on it.

My genetic algorithm is pretty simple, but is fast and scales well to multiple processors. Essentially, each process scrambles the original city order and begins choosing random start and ending points and tests to see if reversing the subsection reduces total distance or not. It compares the distances between cities on the borders of the subsection. The processes are independent, so the program has a higher chance of finding a better solution if there are more processors used. The program runs 8 processes in 2.23728 seconds. 128 ran in 36.2835, but the increased time is likely due to insufficient hardware resources. The total distance was significantly reduced in each case compared to the inital total distance.

2 Program

```
#include <iostream>
#include <mpi.h>
#include <vector>
#include <cmath>
#include <random>
#include <algorithm>

#define MCW MPI_COMM_WORLD
using namespace std;
```

```
class City{
    public:
        long x;
        long y;
        int id;

    City(int id_, int x_, int y_){
        x = x_;
        y = y_;
        id = id_;
    }
};

long double distance(City a, City b){
    return sqrt(((a.x - b.x) * (a.x - b.x)) + ((a.y - b.y) * (a.y - b.y)));
}

int main(int argc, char **argv){
    int rank, size;
    int start, end, maxLength;
    long double cur, next, totalDistance, subDistance, testDistance;
    auto rng = default_random_engine {};

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    srand(rank);

    long double t1,t2;

    vector<City> cities;
    vector<City> sub;

    cities.push_back(City(0,179140,750703));
    cities.push_back(City(1,78270,737081));
    cities.push_back(City(2,577860,689926));
    cities.push_back(City(3,628150,597095));
    cities.push_back(City(4,954030,510314));
    cities.push_back(City(5,837880,811285));
    cities.push_back(City(6,410640,846947));
    cities.push_back(City(7,287850,600161));
    cities.push_back(City(8,270030,494359));
    cities.push_back(City(9,559020,199445));

    cities.push_back(City(10,353930,542989));
```

```
cities.push_back(City(11,515920,497472));
cities.push_back(City(12,648080,470280));
cities.push_back(City(13,594550,968799));
cities.push_back(City(14,386690,907669));
cities.push_back(City(15,93070,395385));
cities.push_back(City(16,93620,313966));
cities.push_back(City(17,426870,39662));
cities.push_back(City(18,437000,139949));
cities.push_back(City(19,789810,488001));

cities.push_back(City(20,749130,575522));
cities.push_back(City(21,481030,286118));
cities.push_back(City(22,670720,392925));
cities.push_back(City(23,273890,892877));
cities.push_back(City(24,138430,562658));
cities.push_back(City(25,85480,465869));
cities.push_back(City(26,775340,220065));
cities.push_back(City(27,862980,312238));
cities.push_back(City(28,155180,263662));
cities.push_back(City(29,274070,74689));

cities.push_back(City(30,333340,456245));
cities.push_back(City(31,822150,399803));
cities.push_back(City(32,158880,612518));
cities.push_back(City(33,815560,707417));
cities.push_back(City(34,678240,709341));
cities.push_back(City(35,394470,679221));
cities.push_back(City(36,631300,846813));
cities.push_back(City(37,528320,824193));
cities.push_back(City(38,666940,845130));
cities.push_back(City(39,298650,816352));

cities.push_back(City(40,243750,745443));
cities.push_back(City(41,220500,654221));
cities.push_back(City(42,338920,381007));
cities.push_back(City(43,313110,201386));
cities.push_back(City(44,856380,564703));
cities.push_back(City(45,549250,565255));
cities.push_back(City(46,537400,604425));
cities.push_back(City(47,502110,435463));
cities.push_back(City(48,498840,590729));
cities.push_back(City(49,482310,571034));

cities.push_back(City(50,416930,765126));
cities.push_back(City(51,418400,638700));
cities.push_back(City(52,374170,695851));
```

```
cities.push_back(City(53,412370,570904));
cities.push_back(City(54,301090,737412));
cities.push_back(City(55,235690,782470));
cities.push_back(City(56,475940,439645));
cities.push_back(City(57,268540,609753));
cities.push_back(City(58,130500,712663));
cities.push_back(City(59,81660,732470));

cities.push_back(City(60,64520,711936));
cities.push_back(City(61,264690,529248));
cities.push_back(City(62,90230,612484));
cities.push_back(City(63,38370,610277));
cities.push_back(City(64,15430,579032));
cities.push_back(City(65,138890,482432));
cities.push_back(City(66,264580,421188));
cities.push_back(City(67,86690,394738));
cities.push_back(City(68,209190,347661));
cities.push_back(City(69,425890,376154));

cities.push_back(City(70,312480,177450));
cities.push_back(City(71,373360,142350));
cities.push_back(City(72,442850,106198));
cities.push_back(City(73,505100,189757));
cities.push_back(City(74,542610,224170));
cities.push_back(City(75,566730,262940));
cities.push_back(City(76,615970,237922));
cities.push_back(City(77,612120,303181));
cities.push_back(City(78,634410,320152));
cities.push_back(City(79,879480,239867));

cities.push_back(City(80,868760,286928));
cities.push_back(City(81,807670,334613));
cities.push_back(City(82,943060,368070));
cities.push_back(City(83,827280,387076));
cities.push_back(City(84,896040,413699));
cities.push_back(City(85,920900,454842));
cities.push_back(City(86,746380,440559));
cities.push_back(City(87,734300,452247));
cities.push_back(City(88,730780,471211));
cities.push_back(City(89,870570,549620));

cities.push_back(City(90,607060,453077));
cities.push_back(City(91,926580,669624));
cities.push_back(City(92,812660,614479));
cities.push_back(City(93,701420,559132));
cities.push_back(City(94,688600,580646));
```

```
cities.push_back(City(95,743800,669521));
cities.push_back(City(96,819700,857004));
cities.push_back(City(97,683690,682649));
cities.push_back(City(98,732680,857362));
cities.push_back(City(99,685760,866857));


if (!rank)
    t1 = MPI_Wtime();

// MPI_Send(&sum,1,MPI_INT,dest,0,MCW);
// MPI_Recv(&incoming,1,MPI_INT,dest,0,MCW,MPI_STATUS_IGNORE);

if (!rank){
    // for (int i = 0; i < cities.size(); i++){
    //     cout << cities[i].id << " " << cities[i].x << " " << cities[i].y << endl;
    // }

    // cout << distance(cities[0],cities[1]) << endl;
}

shuffle(cities.begin(), cities.end(), rng);

for (int i = 0; i < 1000000; i++){
    // generate random start point for subset inverse
    start = rand()%99;

    maxLength = 99-start;

    // generate end point for inverse
    end = start + rand()%maxLength + 1;

    // if better, make the change
    if (start == 0 && end == 99){

    }
    else if (start == 0){
        cur = distance(cities[end], cities[end+1]);
        next = distance(cities[start], cities[end+1]);
        if (cur > next){
            sub = vector<City>(&cities[start],&cities[end]);

            for (int j = 0; j < sub.size(); j++){
                cities[end-j-1] = sub[j];
                // cout << sub[j].id << " ";
            }
```

```
            // cout << endl;
        }
    }
    else if (end == 99){
        cur = distance(cities[start], cities[start-1]);
        next = distance(cities[end], cities[start-1]);
        if (cur > next){
            sub = vector<City>(&cities[start],&cities[end]);

            for (int j = 0; j < sub.size(); j++){
                cities[end-j-1] = sub[j];
                // cout << sub[j].id << " ";
            }
            // cout << endl;

        }
    }
    else{
        // if current distances combined are less than
        // potential distances, don't switch
        cur = distance(cities[start], cities[start-1]) +
            distance(cities[end],cities[end+1]);
        next = distance(cities[start], cities[end+1]) +
            distance(cities[end],cities[start-1]);
        sub = vector<City>(&cities[start],&cities[end]);

        if (cur > next){
            for (int j = 0; j < sub.size(); j++){
                cities[end-j-1] = sub[j];
                // cout << sub[j].id << " ";
            }
            // cout << endl;
        }

        // shuffle(sub.begin(), sub.end(), rng);

    }
}

totalDistance = 0;
for (int i = 1; i < cities.size(); i++){
    totalDistance += distance(cities[i-1],cities[i]);
}

// cout << rank << " total distance: " << totalDistance << endl;
```

```
// rank 0 gathers results (total distances) from other processes

// MPI_Send(&sum,1,MPI_INT,dest,0,MCW);
// MPI_Recv(&incoming,1,MPI_INT,dest,0,MCW,MPI_STATUS_IGNORE);
if (rank){
    MPI_Send(&totalDistance,1,MPI_LONG_DOUBLE,0,0,MCW);
}
else{
    vector<long double> distances;
    long double recv, min;
    min = totalDistance;
    int bestIndex = 0;
    for (int i = 1; i < size; i++){
        MPI_Recv(&recv,1,MPI_LONG_DOUBLE,i,0,MCW,MPI_STATUS_IGNORE);
        distances.push_back(recv);
        if (recv < totalDistance){
            min = recv;
            bestIndex = i;
        }
    }

    cout << rank << " " << totalDistance << endl;

    for (int i = 0; i < size-1; i++){
        cout << i+1 << " " << distances[i] << endl;
    }

    cout << "Best distance: " << min << endl;

    if (bestIndex == 0){
        for (int i = 0; i < cities.size(); i++){
            cout << cities[i].id << " ";
        }
        cout << endl;
    }

    for (int i = 1; i < size; i++){
        int best = 1;
        int notBest = 0;
        if (i == bestIndex)
            MPI_Send(&best,1,MPI_INT,i,0,MCW);
        else
            MPI_Send(&notBest,1,MPI_INT,i,0,MCW);
    }

}
```

```
    if (rank){
        int best;
        MPI_Recv(&best,1,MPI_INT,0,0,MCW,MPI_STATUS_IGNORE);
        // cout << "received " << rank << " " << best << endl;
        if (best){
            cout << "Rank " << rank << endl;
            for (int i = 0; i < cities.size(); i++){
                cout << cities[i].id << " ";
            }
            cout << endl;
        }
    }

    // rank 0 sends messages to other processes and best process prints city sequence

    if (!rank){
        t2 = MPI_Wtime();
        if(!rank)cout << "Total time: " << t2-t1 <<endl;
    }

    MPI_Finalize();

    return 0;
}
```

3 Output

```
$mpic++ traveling_salesman.cpp
$mpirun -np 128 ./a.out

0 2.83931e+07
1 2.83931e+07
2 3.17006e+07
3 3.2609e+07
4 3.49806e+07
5 3.18429e+07
6 3.42976e+07
7 2.9217e+07
8 3.58071e+07
9 3.07358e+07
10 3.60909e+07
11 3.17732e+07
12 3.33281e+07
13 3.26527e+07
```

```
14 3.26318e+07
15 3.13896e+07
16 3.47001e+07
17 3.18011e+07
18 3.30256e+07
19 3.80424e+07
20 3.30466e+07
21 3.62124e+07
22 3.02578e+07
23 3.20497e+07
24 3.59355e+07
25 3.64237e+07
26 3.25056e+07
27 3.26454e+07
28 3.03579e+07
29 3.47146e+07
30 3.12592e+07
31 3.17179e+07
32 3.33905e+07
33 3.16417e+07
34 3.85045e+07
35 3.6135e+07
36 3.33422e+07
37 3.33259e+07
38 3.20874e+07
39 3.40905e+07
40 3.44019e+07
41 3.4002e+07
42 2.80237e+07
43 3.24813e+07
44 3.14614e+07
45 3.6845e+07
46 3.34172e+07
47 3.25004e+07
48 3.56315e+07
49 3.3886e+07
50 3.55032e+07
51 3.06769e+07
52 3.33519e+07
53 3.06771e+07
54 3.95701e+07
55 3.66138e+07
56 2.91151e+07
57 3.34021e+07
58 3.09756e+07
59 3.00478e+07
```

60 3.53607e+07
61 3.14646e+07
62 2.98234e+07
63 3.40617e+07
64 3.26381e+07
65 3.00161e+07
66 3.12411e+07
67 3.35692e+07
68 3.33568e+07
69 3.55242e+07
70 3.56107e+07
71 3.16203e+07
72 3.20239e+07
73 3.2419e+07
74 3.77779e+07
75 3.601e+07
76 3.72036e+07
77 3.51162e+07
78 3.28592e+07
79 3.50077e+07
80 3.13494e+07
81 3.65084e+07
82 3.2716e+07
83 3.58837e+07
84 3.42423e+07
85 3.42863e+07
86 3.06364e+07
87 3.36969e+07
88 3.11277e+07
89 3.37107e+07
90 3.13214e+07
91 3.06589e+07
92 3.36239e+07
93 3.21521e+07
94 3.34773e+07
95 3.75071e+07
96 3.74236e+07
97 3.0512e+07
98 3.24388e+07
99 3.4467e+07
100 3.92957e+07
101 3.18388e+07
102 2.93872e+07
103 3.16887e+07
104 3.06204e+07
105 3.53557e+07

```
106 3.60379e+07
107 3.37853e+07
108 3.43924e+07
109 3.34226e+07
110 3.51032e+07
111 3.12013e+07
112 3.513e+07
113 3.37091e+07
114 3.56857e+07
115 3.18059e+07
116 3.24913e+07
117 3.25921e+07
118 3.98426e+07
119 3.51926e+07
120 3.28923e+07
121 3.22206e+07
122 3.18494e+07
123 3.18386e+07
124 3.29168e+07
125 2.94092e+07
126 3.52944e+07
127 3.6288e+07
Best distance: 2.80237e+07
Total time: 36.2835
Rank 42
59 57 21 74 76 22 33 88 93 34 20 31 69 38 50 83 58 62 54 23 49 29
15 28 67 53 2 24 64 48 39 6 4 1 0 40 55 60 7 61 51 68 32 25 65 66
63 17 43 75 78 30 72 16 70 41 77 56 73 42 9 71 14 94 97 45 8 96 98
44 92 91 84 85 26 86 87 82 95 12 19 81 5 36 3 89 52 13 47 10 37 11
99 35 90 18 27 79 80


$mpic++ traveling_salesman.cpp
$mpirun -np 8 ./a.out

0 2.83931e+07
1 2.83931e+07
2 3.17006e+07
3 3.2609e+07
4 3.49806e+07
5 3.18429e+07
6 3.42976e+07
7 2.9217e+07
Best distance: 2.83931e+07
67 22 86 87 20 70 18 11 89 19 38 78 27 91 44 88 37 6 99 98 96 26
40 47 71 72 17 0 95 12 90 5 33 93 21 74 56 75 94 76 77 83 31 53
```

```
51 50 7 15 49 54 52 57 63 30 3 42 92 81 2 16 65 35 97 14 39 23
62 25 32 64 24 55 58 48 73 79 84 85 28 69 10 59 60 8 1 68 9 4
82 43 66 29 61 45 34 41 46 36 13 80
Total time: 2.23718
```