Assignment 4

Solving TSP using MBF

Parallel Computing(CS 309)

RUCHIR MEHTA
Roll No: 180001044

Date: 23-04-2021

Problem Statement

The problem that is taken into consideration is the Travelling Salesman Problem. “Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?”

In this task, I have implemented a novel algorithm, Mouth Brooding Fish Algorithm, that solves this famous travelling salesman problem. I have implemented both the serial as well as parallel versions of the algorithm. Parallel version is implemented in OpenMP that uses threads. Reference: here.

Code files for serial and parallel implementation: [Attached]

Mouth Brooding Fish Algorithm overview:

This algorithm is inspired by the nature where we find mouth brooders that carry their young ones in its mouth until they are completely mature. This algorithm is based on the behavior and the distance of movement and dispersion of the children around the mother’s mouth. MBF algorithm has 5 controlling parameters. These 5 controlling parameters are the number of population of cichlids (nFish), mother’s source point (SP), the amount of dispersion (Dis), the probability of dispersion (Pdis), and mother’s source point damping (SPdamp). MBF algorithm procedure consists of 4 main parts in order to find the best route results for the TSP.

Effects on cichlids by:
a. The main movements of each cichlid
b. The additional movements of left out cichlids
c. Crossover with roulette wheel selection
d. Shark attack or effects of danger on cichlids movement

Distance between two routes is the sequence of swaps that it takes in going from one route permutation to another route permutation.

1. The main movements of each cichlid
   Basic movements of each cichlid=Sum of all these 4 effects
   a.(due to SP i.e. between 0 and 1)
   i) 1st.Effect = SP × Cichlids.Movements, Cichlids.Movements are the last movements of cichlids.
   ii) in the end of each iteration, SP = SP × SPdamp (due to damping, where SPdamp is between 0.85 and 0.95)
   b.
   i) The second factor of movements is towards the best position of a particular individual cichlid which is found in the past iterations.
   ii) It is controlled by Dis parameter which is between 1 and 2.
   iii) 2nd.Effect = Dis × (Cichlids.Best − Cichlids.Position)

c.
i) tendency of all children to move toward the best possible position of whole cichlids
ii) 3rd.Effect = Dis × (Global.Best − Cichlids.Position)
d.
i) Last factor is nature trends and force
ii) NewN.F.P = 10 × SP × NatureForce.Position(SelectedCells)
4th.Effect = Dis × (NewN.F.P − NatureForce.Position)

2. each child could not move more than Additional surrounding dispersion positive or negative (ASDP or ASDN)
   ASDP = 0.1 × (VarMax − VarMin), ASDN = −ASDP

3. The additional movements of left out cichlids
   a. Left-out cichlids have to face challenges as mother's mouth could not accommodate them
   b. nm = 0.04 × nFish × SP^(−0.431), nm=number of left out cichlids.
   c. Since mother is not caring, they have to move further for survival, so the Additional Surrounding Dispersion positive is multiplied by 4.
   UASDP = 4 × ASDP, UASDN = −UASDP, UASDP and UASDN are the ultra-additional surrounding dispersion positive and negative
   d. LeftCichlids.Position = UASDP ± Cichlids.P(SelectedCells)
   e. Algorithm2

   Crossover with roulette wheel selection. The newly created child will have 0 movement initially.

4. Shark attack or effects of danger on cichlids movement
   a. In nature 4 percent of each colony or population of cichlids are attacked by sharks or other natural threats.
   b. MBF algorithm for the 4 percent of the population uses additional movements named shark attack effect.
   c. n_shark = 0.04 × nFish
   d. Cichlids.NewPosition = SharkAttack × Cichlids.Position,

---

Sequence of steps that are followed in order to create the next generation of Cichlids.

```
//This function creates the nextGeneration from the existing popution
vector<Cichlid> nextGeneration(vector<Cichlid> currentGen,db sp,db natureForce){

    //Firstly the popution is ranked on the basis of the fitness of each Cichlid
    vector<pair<int,db>> popRanked=rankRoutes(currentGen);

    //Global best route is updated
    updateGlobalBest(popRanked,currentGen);
    int n=currentGen.size();

    //Shark attacks are done in order to eliminate the weak population
    vector<int> sharkResults=sharkAttack(popRanked);

    //Parent pool is created where pairs of parents breed to create new Cichlids
    vector<Cichlid> parentpool=parentPool(currentGen,sharkResults);

    //Actual breeding is done here
    vector<Cichlid> children=crossPopulation(parentpool);

    for(int i=0;i<n;i++){

        //Each Cichlid is updated in order to become ready to evolve into the next generation
        children[i].update(globalBestRoute,sp,natureForce);
    }

    //New popution is created and returned from this function
    return children;
}
```

Update function for a Cichlid:

```
void update(Cichlid &globalBestRoute, db sp, db natureForce){

    //Probability that the personal best will make contribution in going to next generation
    db alpha=((double) rand() / (RAND_MAX));
    //Probability that the global best will make contribution in going to next generation
    db beta=((double) rand() / (RAND_MAX));

    //If alpha or beta is >=0.5 then only consider its contribution in deciding the path for next generation
    if(alpha>=0.5)
        alpha=1;
    else
        alpha=0;
    if(beta>=0.5)
        beta=1;
    else
        beta=0;

    //update personal best route
    this->updatePersonalBest();

    //Calculate distance to personal best route and global best route
    Vpll per=this->distanceTo(bestRoute);
    Vpll glo=this->distanceTo(globalBestRoute);
    Vpll toAdd;

    //If none of personal or global wants to make contribution then make it move in the same direction that it was moving
    if(alpha==0 && beta==0){
        this->move(sp,natureForce);
    }else{

        if(alpha==1){
            toAdd.insert(toAdd.end(),per.begin(),per.end());
        }
        if(beta==1){
            toAdd.insert(toAdd.end(),glo.begin(),glo.end());
        }
        this->distanceAdd=toAdd;
        this->move(sp,natureForce);
    }
}
```

Function for Shark Attacks:

```
//This function creates shark Attacks with frequency 90% (assumption) that selects
// the best among the population and leave the weak ones out
vector<int> sharkAttack(vector<pair<int,db>> popRanked){

    vector<int> sharkResults;
    int n=popRanked.size();
    int eliteSize=n/10; //Only 10% survives till next generation
    vector<vector<db>> cum(n,vector<db>(2));
    db sum=0;
    rep(i,n){
        sum+=popRanked[i].second;
        cum[i][0]=sum;
    }
    rep(i,n){
        cum[i][1]=(100*cum[i][0])/sum;
    }
    rep(i,eliteSize){
        sharkResults.pb(popRanked[i].first);
    }

    //Popuation from the remaining 90% is selected randomly to the next generation
    //Using roulette wheel algorithm
    rep(i,n-eliteSize){
        db pick = 100*((double) rand() / (RAND_MAX));
        rep(j,n){
            if(pick <= cum[j][1]){
                sharkResults.pb(popRanked[j].first);
                break;
            }
        }
    }
    return sharkResults;
}
```

Possibility of parallelization:

```
//Paralleliazation happens here
#pragma omp parallel for
for(int i=0;i<n;i++){
    //Each Cichlid is updated in order to become ready
    // to evolve into the next generation
    children[i].update(globalBestRoute,sp,natureForce);
}
```

Here parallelization is done such that as soon as a thread is free it updates one of the chromosome in the population. Here a chromosome means a Cichlid.

This approach uses OpenMP in C++.

Approach that I followed is represented in the figures below.

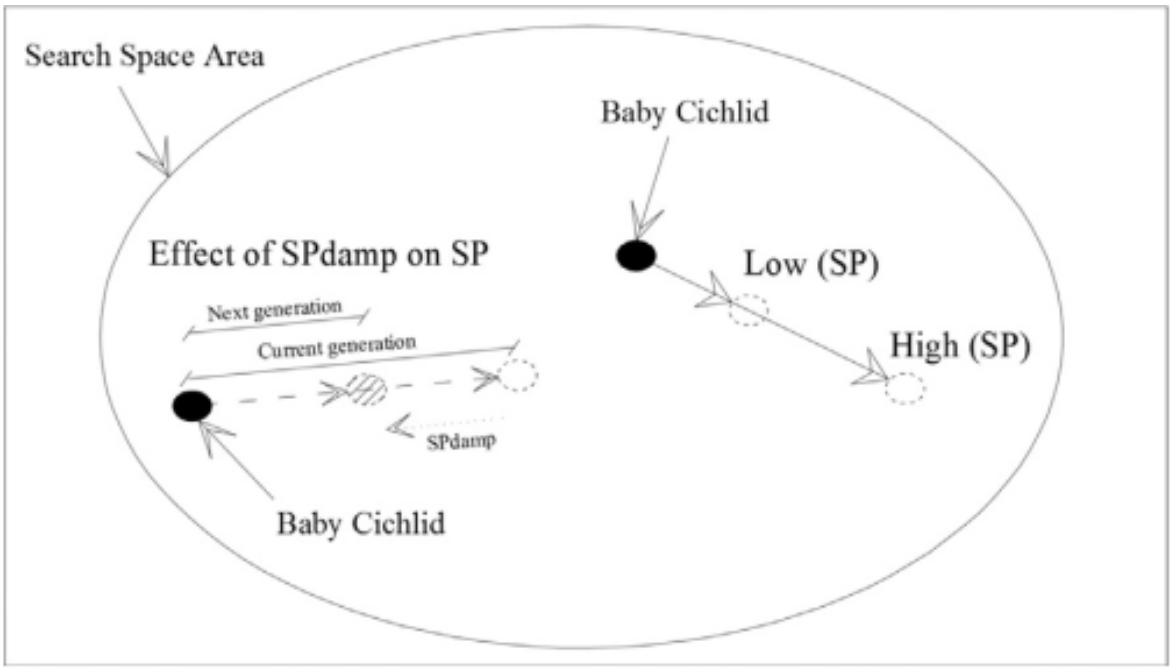

Fig. 3. Effects of SP and SPdamping in cichlids movements.

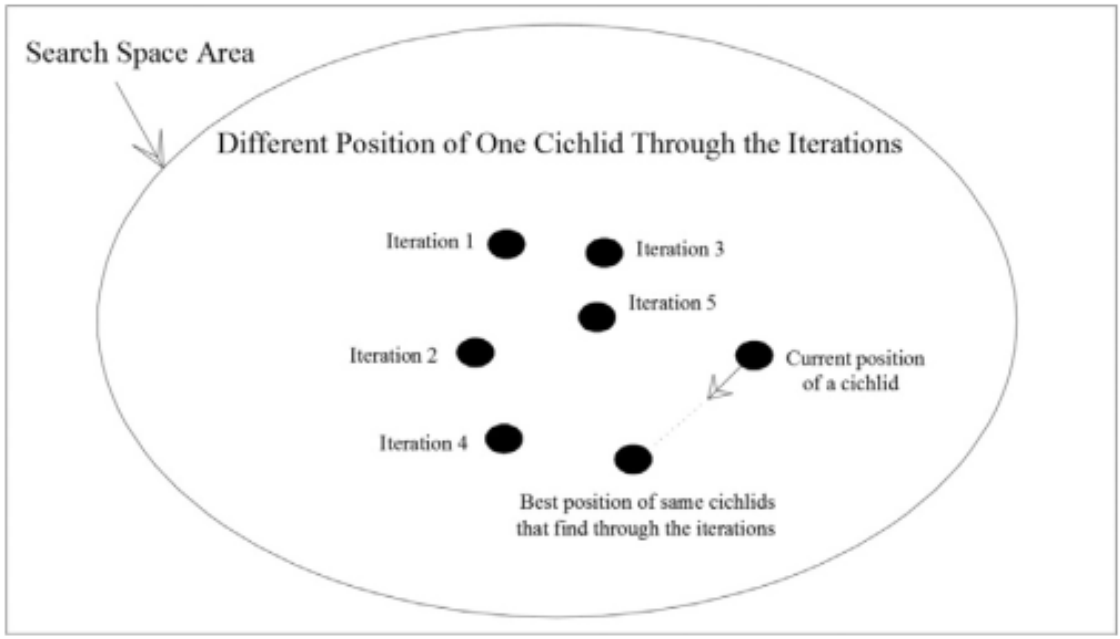

Fig. 4. Effects of best position of each cichlid on movements.

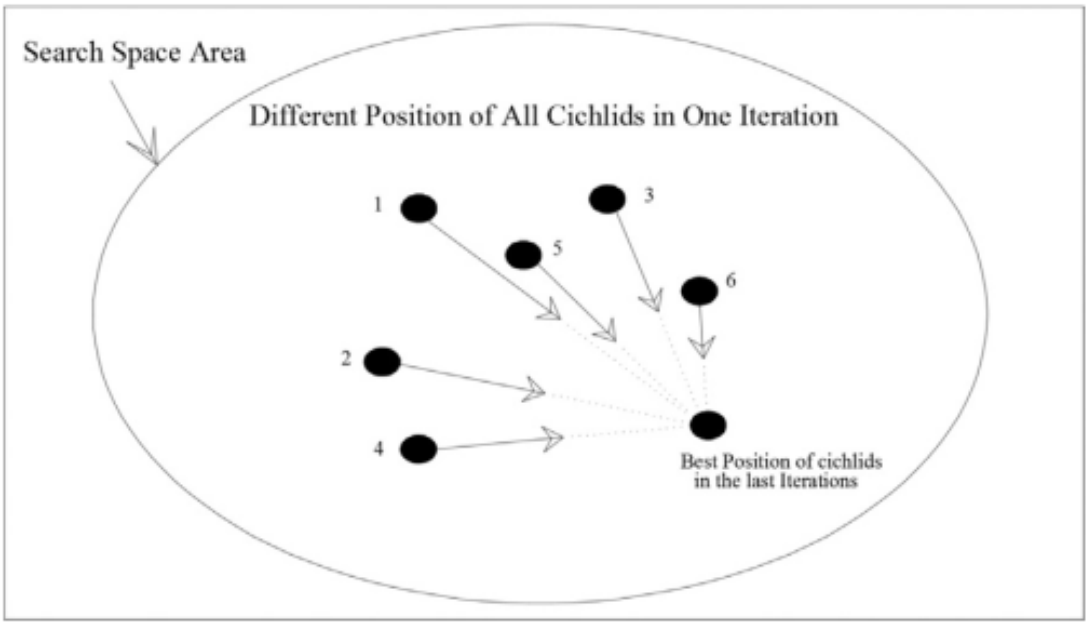

Fig. 5. Effects of the global best position on movements.

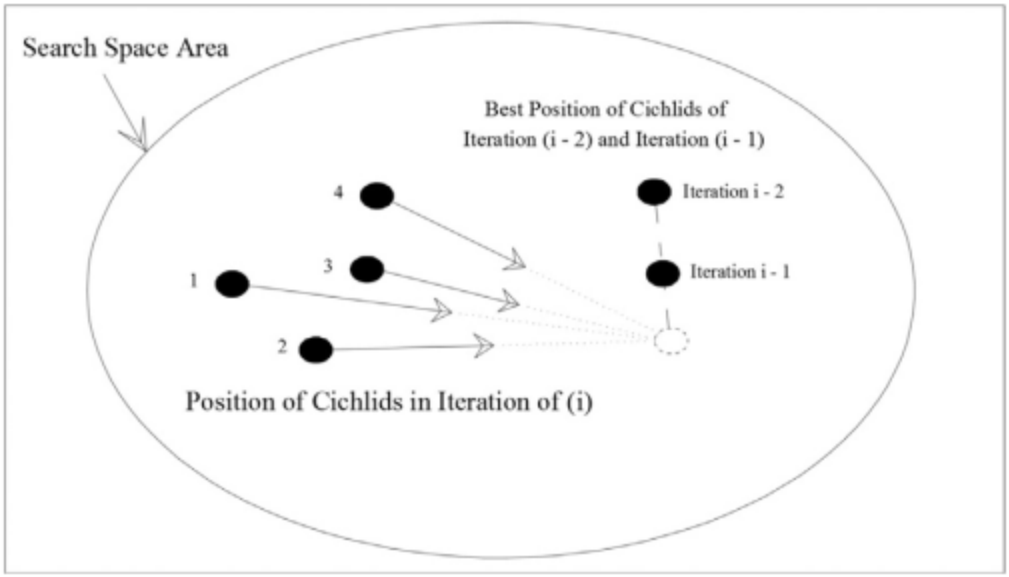

Fig. 6. Effects of nature trend on movements.

Source:
https://www.sciencedirect.com/science/article/abs/pii/S1568494617305744

Time Analysis with Test Cases(Time in seconds):

Serial MBF Algorithm

1. Test-Case

```
Enter number of Cities: 25
Cities:

0: (19, 130)
1: (51, 0)
2: (197, 198)
3: (120, 26)
4: (121, 0)
5: (49, 167)
6: (184, 69)
7: (4, 27)
8: (27, 0)
9: (17, 188)
10: (92, 136)
11: (104, 19)
12: (116, 109)
13: (123, 112)
14: (58, 152)
15: (82, 183)
16: (183, 77)
17: (76, 134)
18: (160, 181)
19: (181, 196)
20: (164, 82)
21: (152, 31)
22: (58, 148)
23: (149, 156)
24: (145, 85)
```

2. Performance towards convergence to global minima

```
Initial Best Distance: 2286.88

0 : 2286.88
1 : 2180.83
2 : 2088.49
3 : 2088.49
4 : 2088.49
```

…..after 1000 generations

```
996 : 885.271
997 : 885.271
998 : 885.271
999 : 885.271

Final Best Distance: 885.271
```

3. Time Taken

```
real    0m7.680s
user    0m5.776s
sys     0m0.024s
```

Parallel MBF Algorithm

4. Test-Case

```
Enter number of Cities: 25
Enter number of threads: 8
Cities:

0: (153, 77)
1: (157, 31)
2: (2, 102)
3: (44, 44)
4: (123, 108)
5: (52, 123)
6: (90, 183)
7: (104, 97)
8: (114, 138)
9: (118, 142)
10: (67, 103)
11: (97, 34)
12: (52, 55)
13: (142, 16)
14: (92, 167)
15: (44, 40)
16: (71, 46)
17: (149, 1)
18: (46, 74)
19: (183, 193)
20: (117, 169)
21: (153, 35)
22: (133, 8)
23: (146, 58)
24: (1, 47)
```

5. Performance towards convergence to global minima

```
Initial Best Distance: 1818.88

0 : 1818.88
1 : 1818.88
2 : 1815.31
3 : 1815.31
```

…..after 1000 generations

```
996 : 930.974
997 : 930.974
998 : 930.974
999 : 930.974

Final Best Distance: 930.974
```

6. Time Taken

```
real    0m8.591s
user    0m44.839s
sys     0m0.218s
```

Observation:

Parallel algorithm show no improvement here, but maybe for larger test cases it will.