Applying the Cuckoo Search to the TSP problem

Ali Zaman[1] and S. Mustafa Ali Zaidi[2]

SZABIST

Karachi, Pakistan

***Abstract* - The firefly algorithm which has been proposed very recently gives very promising results for NP problems, This paper is an attempt to implement the firefly algorithm proposed for QAP and map it on the TSP problem to correct any possible error and to improve the algorithm by making a hybrid version of fir fly algorithm and cuckoo search.**

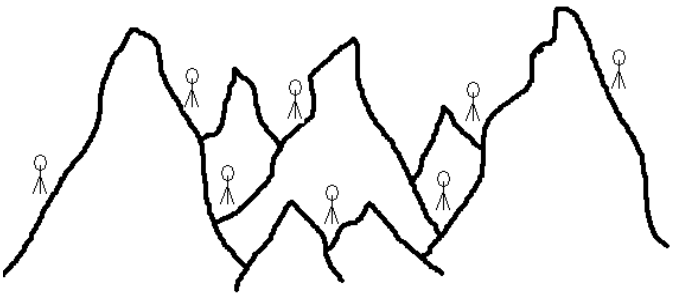
***Keywords* - Swarm Intelligence, Discrete Firefly Algorithm, Cuckoo Search**

I. INTRODUCTION

Swarm intelligence is a relatively new technique which is quite promising for solving problems of minimum and maximum [1]. The core of swarm intelligence lies in the idea of breaking up the solution space. Usually in problems of minimum and maximum the solution space consists of different solutions because there are precise and less precise solutions, and this is because a function is to be fitted and all variables might not be known. So a swarm of agents spread across the solutions space trying to find the most likely pattern. This is better because the probability of getting stuck in local minima is less likely.

Of the many swarm intelligence algorithms proposed, the particle swarm optimization [2] is one of the most important in the context of this paper, for it is very close to the firefly algorithm. In PSO, particles are dispersed across the solution space with different values of some attributes which all particles share. These attributes are based on particle position, particle velocity and direction. The best position is therefore found by each particle. It must be taken into account that the random distribution in the solution space is uniform and that all individual solutions does have an equal probability of showing up. The default random number generator of MATLAB i.e. the Ziggurat algorithm does exactly that [3]. The PSO algorithm also allows the user to define constants so as to allow for the choice to move more either towards the local or global minimum [1]. In the simplest sense the PSO algorithm can be considered as a group of mountain climbers trying to find the point of lowest altitude as shown in the following illustration.

Fig - 1



The PSO is a good method for solving problems; its equation is as follows in equation 1

$$\vec{v}_i(k+1) = \vec{v}_i(k) + c_1\vec{r}_1(k)\cdot\left(\vec{p}_i(k) - \vec{x}_i(k)\right) + c_2\vec{r}_2(k)\cdot\left(\vec{g}(k) - \vec{x}_i(k)\right)$$

Equation - 1

An explanation of the variables is as follows:

k = number of iterations

x(k) = the current position of the particle i in the solution space

v(k) = the current velocity of the particle i

p(k) = the best known position in the solution space known locally to particle i

r1 = randomly generated number between 0 and 1

r2 = randomly generated number between 0 and 1

c1 = user defined constant for inclusion of portion of locally known best solution in calculation of new solution

c2 = user defined constant for inclusion of portion of globally known best solution in calculation of new solution

Nature inspired solutions have also come into the lime light because of the simple fact that of all the solutions that are around us, the solutions proposed by nature are by far the oldest and the most evolved. And it is this fact that makes these solutions probably the most efficient and effective. There are many nature inspired algorithms which have been proposed, the one of most interested in this paper is the Firefly algorithm and the cuckoo search.

II. FIREFLY ALGORITHM

For the purpose of mating, fire flies use their cold flashes. The flashes are a mechanism to attract possible mates and in some cases to devour smaller fire flies [4]. This is the way fire flies communicate with each other. Now there are some non-trivial elements to be considered here. First of all, all fire flies start flying randomly across some space. Then after some while, because of some natural reasons, fire flies start to glow. This glow attracts mates as mentioned above but two non-trivial variables must be considered with this glow, which are: The intensity of the light with which the fire fly glows and the absorption of light in the space in which the fire fly flies. Fire flies move randomly, but the probability of moving towards a more glowing fire is more. From the explanation given above, the following attributes can be extracted, which can be used to model a problem using the firefly algorithm.

- The space within which the fire flies move
- The starting position of fire flies
- The light intensity of the glow
- The light absorption in the selected space
- The number of fire flies
- The duration for which the fire fly flies
- Distance between fire flies

There might be other variables included in the model, but these were the ones proposed in [5]. Thus they are the ones which shall be used and considered in this paper. The algorithm is as follows.

Firefly Algorithm

Objective function $f(\mathbf{x})$, $\mathbf{x} = (x_1, ..., x_d)^T$
Generate initial population of fireflies $\mathbf{x}_i$ $(i = 1, 2, ..., n)$
Light intensity I_i *at* $\mathbf{x}_i$ *is determined by* $f(\mathbf{x}_i)$
Define light absorption coefficient γ
while *(t <MaxGeneration)*
for $i = 1 : n$ *all n fireflies*
  for $j = 1 : i$ *all n fireflies*
    if $(I_j > I_i)$, *Move firefly i towards j in d-dimension;* end if
    Attractiveness varies with distance r via $\exp[-\gamma r]$
    Evaluate new solutions and update light intensity
  end for *j*
end for *i*
Rank the fireflies and find the current best
end while
Postprocess results and visualization

The intensity of light of a fire fly for some other fire fly at distance d can be seen as inversely proportional to the square of the distance. Thus if the intensity at zero distance is Inot the actual intensity would be:

Intensity = Inot / square (distance) .....equation 2

But it isn't just the distance between two fire flies which affects the glow of a fire fly as seen by another. The absorption of light in the selected space also plays a role in the attractiveness. Thus attractiveness becomes:

Attractiveness = Inot / (1 + (light absorption coefficient) (square (distance))).....equation 3

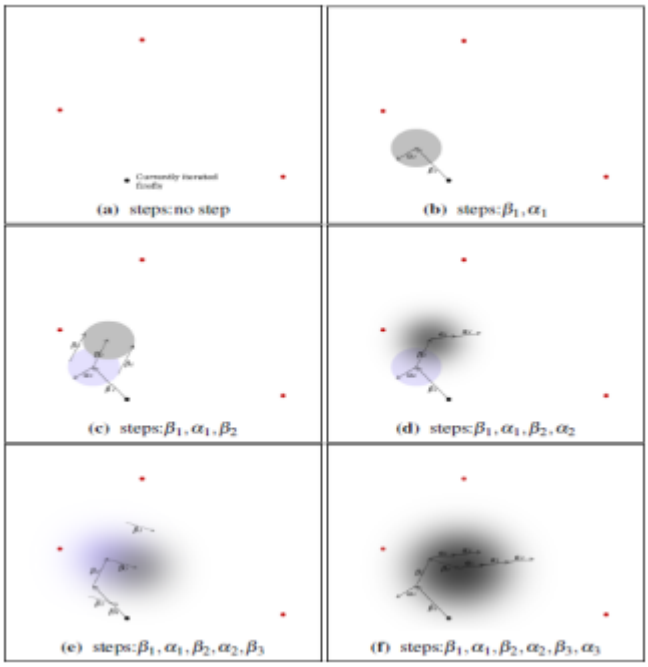
The distance between two fire flies can be considered as simple Euclidian distance in space with n dimensions. Furthermore; it is the best fire fly which moves randomly in the next iteration. The derivation of how the fire flies are to move is given as follows taken from[6].

$$\mathbf{x}^i \leftarrow \mathbf{x}^i + \text{Attractiveness}(I_0, \gamma, \text{Distance}(\mathbf{x}^i, \mathbf{x}^j)) \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2})$$
$$\leftarrow \mathbf{x}^i + \text{Attractiveness}(I_0, \gamma, d_{i,j}) \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2})$$
$$\leftarrow \mathbf{x}^i + I_0 e^{-\gamma d_{i,j}^2} \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2})$$
$$\leftarrow \mathbf{x}^i + \beta \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2})$$
$$\leftarrow (1 - \beta) \cdot \mathbf{x}^i + \beta \cdot \mathbf{x}^j + \alpha \cdot (Random() - \frac{1}{2}),$$

Xi = fire fly with index i
Xj= fire fly with index j
Io = light intensity at zero distance
Gamma = light absorption coefficient
Alpha = randomness coefficient
Beta is simply result after the attractiveness.

Out of these variables; alpha, gamma and Inot are the ones in control of the programmer. Gamma tells us how much a fire fly is to move towards another. And randomness tells us how randomly a fire fly is supposed to move. An example diagram of how the fireflies are supposed to move is shown below [6]. The diagram explain how a fire fly iteratively incorporates the solution of all other fireflies that have a better solution than itself, but it must be noted that if after a step, a fire fly has achieved better solutions than the ones in its surroundings than all other fireflies will move towards it. No matter what may be the case all fireflies will move a bit randomly as well depending on the alpha coefficient given by the user as was discussed above.

Fig - 2



III. Cuckoo Search

This is a fairly simple algorithm which mimics the behaviour of cuckoos, in the sense that bad solutions are discarded so that convergence towards better solutions becomes more probable. It is like a greedy algorithm. It is usually used in combination with other algorithms. So what one basically does in order to implement cuckoo search is that on first use, some algorithm which finds the most probable good solution, sets and then out of that one replaces the subset swarm with randomly selected subset swarm, and then the base convergence algorithm is used to calculate the next swarm movement. The first time it was proposed, it was used with levy flights in [7].

Cuckoo Search via Lévy Flights

begin
  Objective function $f(\mathbf{x})$, $\mathbf{x} = (x_1, ..., x_d)^T$
  Generate initial population of
    n host nests $\mathbf{x}_i$ $(i = 1, 2, ..., n)$
  while *(t <MaxGeneration) or (stop criterion)*
    Get a cuckoo randomly by Lévy flights
      evaluate its quality/fitness F_i
    Choose a nest among n (say, j) randomly
    if $(F_i > F_j)$,
      replace j by the new solution;
    end
    A fraction (p_a) *of worse nests*
      are abandoned and new ones are built;
    Keep the best solutions
      (or nests with quality solutions);
    Rank the solutions and find the current best
  end while
  Postprocess results and visualization
end

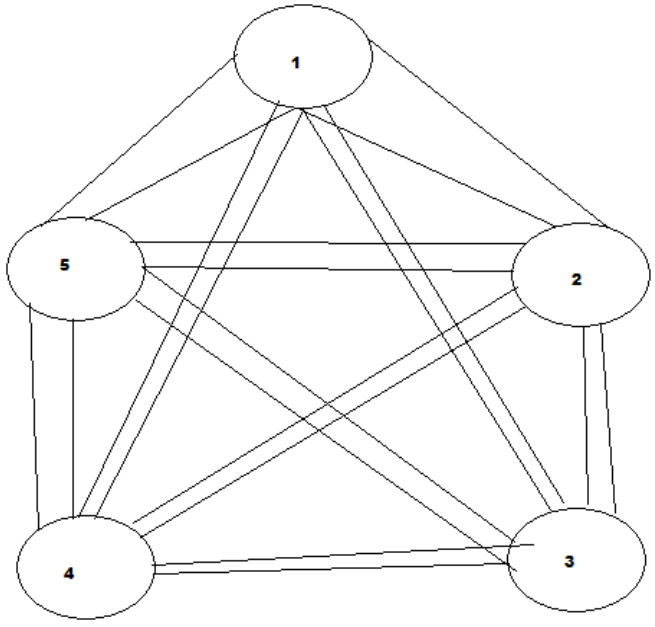
Levy flights is simply a mechanism of finding orthogonal solutions in a random space and it has been shown that this way of searching can be much faster.

IV. The Travelling Salesman Problem

It is one of the oldest problem discussed in computer science. It is an NP hard problem, thus giving a deterministic solution is not very much possible for all instances of the problems as yet. The problem is quite simple in terms of its semantics. It states that there are cities which are interconnected and a travelling salesman who is supposed to travel in such a way that he visits each city only once and finds the shortest path to do so. In this paper all cities are connected with all other cities. The problem lies in the class of discrete problems. For the purpose of implementation, the programming done for this paper and the graph was implemented as a 2D matrix whose main diagonal was ignored in solution construction so as to avoid a path from a city to itself, a pair of coordinates on the matrix would give the distance between two nodes in the graph. The main advantage of following this procedure is that it makes it very easy to make a fully connected graph with random weights. All has to do is write a one line MATLAB statement and the graph is instantly made, afterwards, all is to do is to save that graph so that only it is used for comparison. The simple MATLAB statement is as follows.

Graph = ceil (rand(5)*10);

The statement would make a matrix equivalent to the graph shown in the following diagram with random weights between 1 and 10.




```
>> ceil(rand(5)*10)

ans =

     9     1     2     2     7
    10     3    10     5     1
     2     6    10    10     9
    10    10     5     8    10
     7    10     9    10     7
```

Fig - 3

Consider a permutation with numbers from 1 to 5 in a non-repeating way e.g. (1 3 2 5 4). To find the distance of this path in the graph all one has to do is perform some additions. In the above graph one can simply follow the following sequence of additions

ans (1 , 3) + ans (3 , 2) + ans (2 , 5) + ans (5 , 4) = 2 + 6 + 1 + 10 = 19

the numbers on the diagonals mean nothing. Thus they can be ignored altogether. The graph is directed in the sense that path from one node to another is different depending on the direction. e.g. the distance from 1 to 2 is 1 while that from 2 to 1 is 10.

V. The Problem With The Firefly Algorithm & The Travelling Salesman Problem

The Firefly Algorithm as proposed originally caters for solutions in which we are performing some kind of regression. It can be seen that it assumes that fireflies consist of random values that have continuous values. Secondly when one firefly is supposed to move towards another the thing which is done is that the difference between them is lessened, where difference is the value one gets by subtracting one value from another.

The TSP as specified above is a different problem than the one this algorithm is designed to tackle. The TSP is a discrete problem which consists of graphs. The solution is supposed to be a sequence which leads one through the whole graph without repeating a node. Secondly, since the solution is a path, therefore when it is said that one path is closer to another or farther from another, the entities being used to do this comparison are the node placements in the specific path sequences. Hamming distance is used for this purpose as proposed in [7].

The firefly for this type of problems was found to be proposed in [7]. The complicated part is the beta step of this algorithm. What it is supposed to do is that on the basis of the number of nodes which are different in two solutions, it is supposed to make the two similar by calculating a probability and then on the basis of that probability a number of elements are selected randomly that are to be changed. But there was a problem with the proposed solution, which becomes clear when we see the following example. Consider the following two permutations:

[4-9-3-7-6-8-2-1-5].....sequence 1

[4-1-3-2-6-5-9-7-8].....sequence 2

The proposed beta step suggests that we randomly compare elements and change one element in accordance with the probability formula proposed. So after the application of the beta step, the sequence becomes

[4-1-3-7-6-8-2-9-5].....resultant sequence

But if it is considered that the second sequence towards which the first sequence was supposed to be moved was:

[4-1-3-2-6-5-7-9-8]

The resultant sequence, instead of being just one hamming distance unit closer to the second sequence, it is two hamming distance units closer. This introduces the probability of giving different amount of motion towards better solutions for different types of problems and random solutions. This problem was corrected in the code produced for this paper.

VI. BASIC FIREFLY ALGORITHM FOR TSP

```
function
x=fireFlyAlgorithm(graph,iterations,lightAbsorption,lightIntensity,numberOfFireFlies,randomSte
pSize)
    permutationSize=length(graph);
    distances=zeros(1,numberOfFireFlies);
    fireFlies={};
    for i=1:1:numberOfFireFlies
        fireFlies{1,i}=shuffleMatrix(1:1:permutationSize);
    end
    minimum={};
    for i=1:1:iterations


        for j=1:1:numberOfFireFlies
            distances(1,j)=calculateDistance(fireFlies{1,j},graph);
        end

        [c,column]=min(distances);
        minimum{1,1}=distances(1,column);
        minimum{1,2}=fireFlies{1,column};

        for k=1:1:numberOfFireFlies
            for l=1:1:numberOfFireFlies
                if
((calculateDistance(fireFlies{1,k},graph))<(calculateDistance(fireFlies{1,l},graph)))
                    y=betaStep(fireFlies{1,l},fireFlies{1,k},lightAbsorption,lightIntensity);
                    y=alphaStep(y,randomStepSize);
                    fireFlies{1,l}=y;
                end
            end
        end

        fireFlies{1,column}=shuffleMatrix(1:1:permutationSize);
    end
    x=minimum;
end
```

The first thing which comes to mind is the question, what this shuffle matrix function does. This function simply takes a matrix and shuffles it in a random manner by first making as many stacks as there are elements in the matrix, then discrete random numbers are generated in order to select the stack on which an element is to be put. In the end, all the stacks are put on top of each other. The function is called shuffle matrix because it creates random sequences in a similar manner used to shuffle cards.

A. Beta Step

Now the beta step shall be looked at in detail. The algorithm for the beta step is given as follows:

```
function x=betaStep(permutationOne,permutationTwo,lightAbsorption,lightIntensity)
    x=zeros(1,length(permutationOne));
    indexTwo=getLocation(permutationTwo);
    differenceMatrix=permutationOne-permutationTwo;
    binaryRepresentation=convertToBinary(differenceMatrix);
    hammingDistance=sum(binaryRepresentation);
    %calculate attractiveness and find out number of elements to be selected from matrix2%
    attractiveness=lightIntensity/(1+(lightAbsorption*(hammingDistance^2)));
    forcedRandomness=ceil(rand(1,1)*9);
    elementsToBeSelected=0;
    if (forcedRandomness<=3)
        elementsToBeSelected=floor(attractiveness*hammingDistance);
    elseif (forcedRandomness<=6)
        elementsToBeSelected=round(attractiveness*hammingDistance);
    elseif (forcedRandomness<=9)
        elementsToBeSelected=ceil(attractiveness*hammingDistance);
    end
    classifiedElements=elementsToChangeAndLeave(permutationTwo,elementsToBeSelected);
    switchingElements=classifiedElements{1,1};
    leftElements=classifiedElements{1,2};
    binaryRepresentation=adjustBinaryMatrix(indexTwo,switchingElements,binaryRepresentation);
    x=changePermutation(permutationOne,indexTwo,switchingElements);
    differenceMatrix=x-permutationTwo;
    binaryCheck=convertToBinary(differenceMatrix);
    binaryError=binaryCheck-binaryRepresentation;
    x=correctPermutation(x,indexTwo,leftElements,binaryError);
```

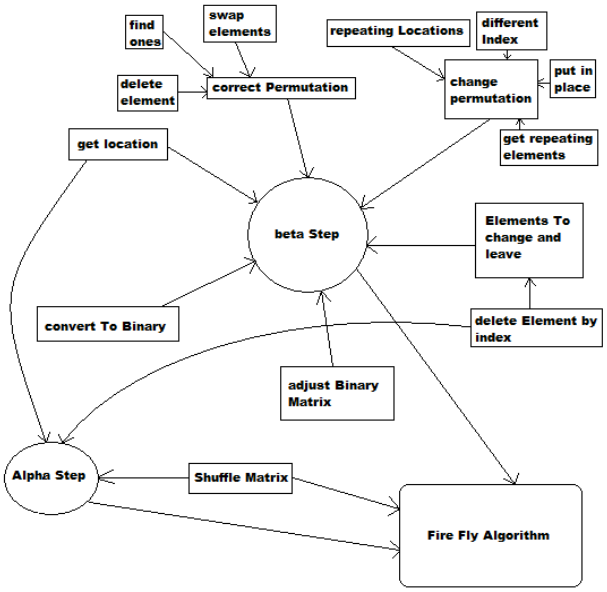
The different components on which the beta Step depends

B. Alpha Step

The alpha step simply swaps a given number of elements from random locations.

```
function x=alphaStep(permutation,stepSize)
    temp=permutation;
    selectedElements=zeros(1,stepSize);
    for i=1:1:stepSize
        randomLocation=ceil(rand(1,1)*length(temp));
        selectedElements(1,i)=temp(1,randomLocation);
        temp=deleteElementByIndex(temp,randomLocation);
    end
    locations=getLocation(permutation);
    selectedLocations=zeros(1,stepSize);
    for i=1:1:stepSize
        selectedLocations(1,i)=locations(1,selectedElements(1,i));
    end
    randomizedElements=shuffleMatrix(selectedElements);
    for i=1:1:stepSize
        permutation(1,selectedLocations(1,i))=randomizedElements(1,i);
    end
    x=permutation;
end
```

The interdependency diagram of the whole firefly algorithm becomes the following:



Some components have been left out.

VII. How To Apply The Cuckoo Search To TSP

The cuckoo search algorithm in itself is very simple. Cuckoos have a habit of laying their eggs in nests of other birds. Not only cuckoos do that but they also throw away some of the eggs of the host bird. This is done so that its own eggs have a better chance of being nourished. When it is applied to problems, the only thing used in this behavior is that out of the solutions proposed, the bad solutions are replaced by new random solutions.

Thus in itself the cuckoo search is a fairly simple algorithm but it must be integrated with another algorithm so that it becomes effective. In [7] it is used with Levy flights which is a distribution which tends to select only particular random numbers, converting Levy flights to TSP problem would have been a problem in itself. Thus what was needed was an algorithm which works for discrete problems on which the cuckoo search could be applied. The cuckoo search has been proposed for the Nurse scheduling problem in combination with Evolutionary computation algorithm [9]. But EA is old compared to the firefly algorithm which has been proposed very recently, so for this paper a new hybrid algorithm was proposed and implemented, namely the cuckoo search with the firefly algorithm. The only thing which makes this new algorithm different from the older version of firefly algorithm is that in the original firefly algorithm the best firefly moves randomly, whereas in this hybrid version the worst fireflies are moved randomly, so this also introduces a kind of greediness in the firefly algorithm. The new proposed algorithm works as follows.

```
function x=fireFlyWithCuckoo(graph,iterations,lightAbsorption,lightIntensity,numberOfFireFlies,randomStepSize,fractionSize)
  permutationSize=length(graph);
  distances=zeros(1,numberOfFireFlies);
  fireFlies={};
  for i=1:1:numberOfFireFlies
    fireFlies{1,i}=shuffleMatrix(1:1:permutationSize);
  end
  minimum={};
  for i=1:1:iterations


    for k=1:1:numberOfFireFlies
      for l=1:1:numberOfFireFlies
        if ((calculateDistance(fireFlies{1,k},graph))<(calculateDistance(fireFlies{1,l},graph)))
          fireFlies{1,l}=betaStep(fireFlies{1,l},fireFlies{1,k},lightAbsorption,lightIntensity);
          fireFlies{1,l}=alphaStep(fireFlies{1,l},randomStepSize);
        end
      end
    end

    fireFlies=mergeSort(fireFlies,graph);
    numberToDiscard=ceil(length(fireFlies)/fractionSize);
    for m=1:1:numberToDiscard
      fireFlies{1,m}=shuffleMatrix(1:1:permutationSize);
    end

      minimum{1,1}=calculateDistance(fireFlies{1,length(fireFlies)},graph);
      minimum{1,2}=fireFlies{1,length(fireFlies)};


  end
  x=minimum;
end
```

The only difference as can be seen clearly is that a merge sort has been used in order to first sort the different solutions and then a fraction of the number of fireflies is replaced with random solutions using the shuffle matrix algorithm.

VIII. Results

A. *Graph*

Before going into the result first the environment of the experiment will be elaborated. The TSP graph used was one consisting of 50 cities. A matrix was used as explained above.

B. *Iterations used*

500 iterations were used in order to converge towards a solution.

C. *Light absorption*

A light absorption coefficient of 0.3 was used.

D. *Light Intensity*

Light intensity of 1 was used.

E. *Number of fire flies*

A firefly in this algorithm represents a random sequence instantiated in the beginning. This is a combination of

numbers representing a path in the graph. 40 fireflies were selected.

F. Random Step size

The random step size specifies how many elements to shuffle after the beta step has been taken. In other words it is the input which directs the alpha step. A random step size of 15 was used in this experiment.

G. Fraction Size

Half of the solutions were replaced with random sequences in the cuckoo search version. A fraction size of 2 was used. The more the size the lesser the number of bad solutions replaced.

X=Minimum distances achieved using the Fire Fly algorithm

Y= Minimum distances achieved using The Cuckoo search with the fire fly algorithm

TABLE I

	X	Y
	214	180
	211	187
	217	178
	219	186
	215	184
	216	183
	211	182
	211	179
	215	183
	208	172
Average	213.7	181.4
Difference 32.3		

These mere 32.3 units can mean a lot in context of the resources which the distances represent.

IX. CONCLUSION & FUTURE WORK

- It is clear that the firefly algorithm in collaboration with the cuckoo search gives better results compared to the standalone version of the firefly algorithm. But here are many things which could be done to improve this algorithm further. A list of possibilities is given below:
- Instead of just making nodes closer, pairs of nodes could have been chosen representing arcs.
- It would be a good idea to first convert the graph into a Euclidian graph so that geometric properties can be used as well, that would make the firefly algorithm more natural and may produce even better results.
- In the Euclidian graph it would be easier to use levy flights [10].

REFERENCES

[1] George I. Evers. “An Automatic Regrouping Mechanism To Deal With Stagnation In Particle Swarm Optimization”, M.S Thesis, University of Texas, 2009.

[2] Marco Dorigo et al. “Particle swarm optimization”. http://www.scholarpedia.org/article/Particle_swarm_optimization, November2008.

[3] George Marsaglia, Wai Wan Tsang, “The Ziggurat Method for Generating Random Variables”, Journal of statistical software, Vol. 5, Issue 8, Oct 2000.

[4] Encyclopedia Britannica. Firefly. http://www.britannica.com/EBchecked/topic/207935/FireFly

[5] Xin-She Yang, ”FireFly Algorithms for Multimodal Optimization”, Stochastic Algorithms: Foundations and applications, 2009

[6] Karel Durkota, “Implementation of a Discrete FireFly Algorithm for the QAP Problem within the SEAGE Framework”, Bachelors thesis, Czech Technical university in Prague, 2011

[7] Xin-She Yang Deb, S. “Cuckoo Search via Levy Flights”, Nature & Biologically Inspired Computing, 2009

[8] Greg Randall, “Levy flights”, http://ocw.mit.edu/courses/mathematics/18-366-random-walks-and-diffusion-fall-2006/study-materials/lecture12.pdf, 2003

[9] Lim Huai Tien, Razamin Ramli, “Rescent Advancement of Nurse Scheduling Models and A Potential Path”, Proceedings of the 6th IMT-GT Conference on Mathematics, Statistics and its Applications, 2010

[10] James Decay, ”Sharks hunt vi Levy Flights” http://physicsworld.com/cws/article/news/42899, 2010