1

Implementation of the Firefly and Ant Colony optimization algorithm on NP-hard Bin Packing Problem (1D packing)

Abhirupa Mitra 17BCE0437, Nikhil Anand 17BCB0053

***Abstract* — Bin Packing Problem is categorized as a NP - Hard Optimization Problem. This classic problem aims at reducing the no. of bins used for storing a set of items, exactly once with different weights. The algorithm of bin packing can be utilized in many real world applications such as transportation planning, container loading, resource allocation, cargo planes and ships. Traditional methods such as First-fit and Best-Fit algorithms are usually implemented to solve this problem. In this project we are going to use a bio-inspired algorithms like firefly and ant colony optimization techniques. Firefly algorithm is inspired by the flashing behavior of fireflies. The ant colony optimization is based on the behavior of ants seeking a path between their colony and source food. These algorithms are metaheuristic algorithms which aim at further reducing bin space wastage and execution time when compared with the traditional algorithms. Bio-inspired algorithmic usage has made a significant impact in many fields such as Computer Science, Electronics, Mechanical Engineering and Artificial Intelligence. The field of nature-inspired computing and optimization techniques have helped solve a diverse range of optimization problems in the field of science and technology. This project will implement the Firefly and the Ant Colony Optimization Algorithms for 1D Bin Packing.**

***Index Terms*— Bin packing, ant colony optimization, firefly, meta heuristic, firefly algorithm,**

I. Introduction

Bin Packing

Bin packing have been categorized as a NP- hard combinatorial optimization problem. NP-hardness(non-deterministic polynomial-time hardness), in computational theory of an algorithm for solving it can be translated into an algorithm fit for solving any NP problem.

In classical bin packing problem, a set of items (Say, n) of different sizes or weights has to be packed into bins of a limited capacity. Optimization will require us to find out the minimum no. of bins that can be used for the packing. Given a set of bins with the same size and a list of items with sizes to pack, find an integer number of bins and a -partition of the set such that for all A solution is *optimal* if it has minimal . The B -value for an optimal solution is denoted **OPT** below. A possible Integer Linear Programming formulation of the problem is:

$$\text{minimize } B = \sum_{i=1}^{n} y_i$$

Bin

$$\text{subject to } B \geq 1,$$

$$\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \forall i \in \{1,\ldots,n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in \{1,\ldots,n\}$$

$$y_i \in \{0,1\}, \qquad \forall i \in \{1,\ldots,n\}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in \{1,\ldots,n\}\, \forall j \in \{1,\ldots,n\}$$

where $y_i = 1$ if bin i is used and $x_{ij} = 1$ if item j is put into bin i.

packing algorithm is efficiently used in solving a variety of manufacturing problems such as cutting stock problems and waste minimization resulting in cost minimization.

Nature is viewed as a great and immense source of inspiration for solving hard and complex problems. It reflects the tendency of biological creatures to adapt to natural conditions. It always attempts to figure out an optimal solution to solve its problem, thus maintaining a perfect balance among its components. Nature inspired algorithms are meta-heuristics that mimic nature to solve optimization problems opening a new era in computation. Optimization means finding the best possible or the most optimum solution. Nature inspired algorithm have the ability to describe and solve complex relationship based problems. Each phenomenon occurring in nature and the simultaneous response of a living creature to it acts as a subtle example of optimal strategy and complex interaction, adaption to constraining conditions and thus balancing the ecosystem.

Classical Bin Packing Problem Statement:

Given n items with weights w_i and bin capacity c, assign the number of bins so that total weight of items in each bin does not exceed c.

Models and Bounds:

The generalized bin packing problem can be formulated as upper bound GBPP and lower bound GBPP.

Let *I* denote the set of items that has to be place in the bins and w_i and p_i be the weight and the profit of item I $\in$ *I*. Let $I_C \subseteq$ *I.* the subset of compulsory items and $I._{NC}$ = *I.* \ $I._C$ the subset of non-compulsory items. Let J denote the set of available bins and T be the set of bin types. For any bin j ∈ J, let σ(j) = t ∈ T be the type t of bin j. Define, for each bin type t ∈T, the minimum L_t and the maximum U_t number of bins of that type that may be selected, as well as the cost C_t and the volume W_t of the bin. Finally, denote U ≤P ∈T U_t the total number of available bins of any type. The item-to-bin accommodation rules of the GBPP are stated as follows:

- All items in I^C must be loaded
- For all used bins, the sum of the volumes of the items loaded into a bin must be less than or equal to the bin volume

1

• The number of bins used for each type t ∈ T must be within the lower and upper availability limits L_t and U_t
• The total number of used bins cannot exceed the total number of available bins.

Traditional Algorithms Used to solve the Bin Packing Problem:

1. First-Fit Algorithm: This algorithm assumes that the items are arbitrarily arranged and starts putting in items into bins. Each item is assigned the the lowest indexed bin. If an item does not fit into a particular bin, then a new bin is introduced. The time complexity of this algorithm is O(n).
2. Best-Fit Algorithm: Best fit algorithm improves upon the first fit algorithm, by assigning the current item into a bin which has the minimum residual capacity. Contrary to that the worst fit selects a partially filled bin having the largest residual capacity.

II. LITERATURE REVIEW

The GBPP is a novel packing problem recently introduced by Baldietal. [2012a]. In their paper, the authors propose two models and preliminary bounds. A branch and-price method and beam search heuristics have been proposed in [Baldi et al., 2012b]. The stochastic variant of the problem has been studied by Perboli et al. [2012].. The BPP is the simplest mono-dimensional bin packing problem, introduced by Ullman [1971]. Johnson proposed some preliminary algorithms. In particular, in [Johnson, 1973a], he proposed the Next Fit (NF) algorithm, he proposed the First Fit (FF), Best Fit (BF), First Fit Decreasing (FFD), and Best Fit Decreasing (BFD). Basing their studies on the work of Johnson, algorithms in the years. Yao [1980] presented the Refined First Fit algorithm, with performance ratio 5/3. Afterward, van Vliet [1992] increased the lower bound to 1.54014. Lee and Lee [1985] presented a family of bounded space algorithms, named Harmonic M. To the best of our knowledge the best result to date is due to Seiden [2002] who proposed the Harmonic++ algorithm with performance ratio at most 1.58889. Preliminary bounds to the BPP were proposed by Martello and Toth [1990]. New lower bounds were developed by Fekete and Schepers [2001] by means of dual feasible functions. Epsteinand van Stee [2005] studied the problem by means of resource augmentation. Kouakou et al. [2005] studied the problem with respect to the differential competitivity ratio. Finally, György et al. [2010] studied on-line Sequential Bin Packing Problem. Epstein and Levin [2012] have recently designed an asymptotic fully polynomial time approximation scheme for this problem. Seiden[2000] proposed an optimal on-line algorithm for the bounded space(i.e., the number of open bins is constant) problem. Seiden et al. [2003] proposed improved bounds but with two bin sizes only. Alves and Valério de Carvalho [2007] first proposed an improved column generation technique trying to solve the VSBPP to optimality. An on-line variant of the VSBPP was introduced by Zhang [1997] Kang and Park [2003] studied the problem assuming that the cost of the unit size of each bin does not increase as the bin size increases. The authors proposed two greedy algorithms and computed their asymptotic worst case ratio under three assumptions that the sizes of items and bins are divisible (i.e., the succeeding item (bin) exactly divides the previous item (bin)), the sizes of bins are divisible, and the sizes of bins are not divisible. The authors proved that both algorithms yield an asymptotic worst case ratio equal to 1 (i.e., the two algorithms are optimal) under assumption 1, equal to 11/9 under assumption 2, and equal to 3/2 under assumption 3. For Epstein and Levin [2008] designed an asymptotic polynomial time approximation scheme. Correia et al. [2008] proposed a formulation that explicitly includes the bin volumes occupied by the corresponding packings, together with a series of valid inequalities improving the quality of the lower bounds obtained from the linear relaxation of the proposed model. Approximation algorithms have been proposed by Haouari and Serairi [2009] and Hemmelmayr et al. [2012]. Recently, Bettinelli et al. [2010] introduced a branch-and-price algorithm for the resolution o.f a variant of the VCSBPP with the addition of filling constraints. The latest work dealing with exact methods to solve VCSBPP is by Haouari and Serairi [2011].The OKP has been studied by Iwama and Taketomi [2002], Iwama and Zhang [2007,2010].

III. ALGORITHMS

Ant Colony Optimization:

In computer science and in optimization problems, the ant colony optimization algorithm is a technique which can be used to finding good paths through graphs.

Some species of ants travel randomly searching for food, and on finding it, return to their colony leaving a pheromone trail on their way back. If other ants locate this trail, they stop wandering randomly but follow the trail retuning and reinforcing if they eventually find food. With time the pheromone starts evaporating, thus reducing its attractive strength. The greater is the time taken by an ant to travel the path and come back again, the greater amount of pheromone is evaporated. A shorter path gets marched over more frequently, hence the pheromone density is high. The phenomenon of pheromone evaporation prevent ants to converge to a locally optimal solution.
The overall result is that when an ant finds/locates a good/shorter path, more ants are likely to follow that path, thus increasing pheromone content, thus resulting in all ants to follow this path.

Framework for Ant Colony Optimization

```
WHILE termination conditions not met DO
  ScheduleActivities
    AntBasedSolutionConstruction()
    PheromoneUpdate()
    DaemonActions() {optional}
  END ScheduleActivities
ENDWHILE
```

Firefly Algorithm

Firefly algorithm, developed by Xin-She Yang in 2008, is inspired by the light attenuation over the distance and fireflies' mutual attraction, rather than by the phenomenon of the fireflies' light flashing. Algorithm considers what each firefly observes at the point of its position, when trying to move to a greater light-source, than is his own. Some of the assumptions taken in the algorithm are:

1. **All fireflies are unisexual**, so that any individual firefly will be attracted to all other fireflies;
2. **Attractiveness is proportional to their brightness**, and for any two fireflies, the less bright one will be attracted

by (and thus move towards) the brighter one; however, the intensity (apparent brightness) decrease as their mutual distance increases;

3. **If there are no fireflies brighter than a given firefly, it will move randomly.**

Framework for Firefly Algorithm

```
GENERATE AN INITIAL POPULATION OR FIREFLIES
AND FORMULATE LIGHT INTENSITY
DEFINE ABSORPTION COEFFICIENT
TRAVERSE FROM 1 TO NTH FIREFLY IN A NESTED
LOOP FOR 2 LOOP VARIABLES i AND j
      CHECK FOR ATTRACTION AND MOVE FIREFLY
      ACCORDINGLY
POST PROCESS THE RESULTS AND VISUALIZATION
END
```

IV. Traditional Methods

First Fit: Arrange the items in an order and insert it. Take a new bin if the new item cannot be accommodated in the first one.

assignBins()

```
bins ← 1
i ← 0
tw ← 0;
while i < num do
      tw ← tw + items[i]
      if tw > cap do
         bins ← bins + 1
         tw ← 0
      else
         i ← i + 1
return bins
```

Best Fit: Insert the item in the bin with the minimum residual capacity

assignBins()

```
i ← 0
bins ← 0
for k ← 0 upto num do
      binlist[k] ← cap;
      while i < num do
            sortbins()
            for j ← 0 upto num do
            if items[i] <= binlist[j] do
                  binlist[j] ← items[i]
                  break
            i ← i + 1
            for i←0 upto num do
                  if binlist[i] != cap do
                        bins ← bins + 1
return bins
```

V. Ant Colony

Conditions:

The dimensions of the bins is such that it can handle any number of boxes until the total weight of the boxes in it does not exceed the carrying capacity.

The boxes are kept linearly from the bins.

AntBasedSolutionConstruction()

```
      int i= int(random()*n)
      int c;
    int totalwt=0;
    while(bins[binN]>w[i])//Solution     for
first bin by 1 ant
    {
        if(bins[binN]<w[i])//Size    exceeds
bin capacity
            break;
        bins[binN]=bins[binN]-w[i];
        totalwt+=w[i];
        w[i]=-1;
        c++;//Counting   no.   of   objects
inserted into bin
        do
        {
            i=(int)(Math.random()*n);
        }while(w[i]==-1);
     }
        Update pheromone content
```

PheromoneUpdate()

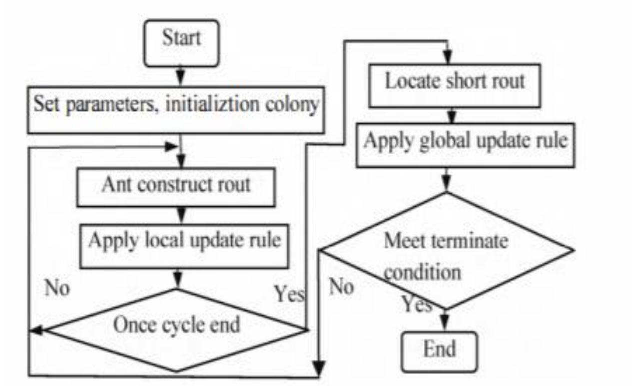
```
      double temp=0;
    temp=totalwt/(double)binN; //Calculating
pheromone update for each ant
    if (temp>pheromone[binN])
    {    //Taking pheromone update from the
best ant solution

            if (visited[binN]==true)
            {
                pheromone                +=
Math.pow(graph[i][l], alpha) * Math.pow(1.0
/ graph[i][l],
                  beta);         //Updating
pheromone
            }
    }
```

AntColonyOptimization()

```
      int trailSize = pheromone.length();
    double            length              =
graph[pheromone[trailSize                 -
1]][pheromone[0]];
    for (int i = 0; i < trailSize - 1; i++)
    {
      length                             +=
graph[pheromone[i]][pheromone[i + 1]];
    }
    return length;
```

FLOW DIAGRAM:



VI. FIREFLY ALGORITHM

Conditions:
The dimensions of the bins is such that it can handle any number of boxes until the total weight of the boxes in it does not exceed the carrying capacity.
The boxes are kept linearly from the bins.

Factors:
Attractiveness: The attractiveness here is the residual space that a box will leave in the bin once it has been kept in it. In other words, the less the weight of the box, the more the attractiveness
Distance: The distance is referred as the difference in position with the box that is attracting with the current box taken into account
Favorability: The net result obtained by subtracting the attractiveness over the distance. As the distance from the current attracting body and the body that is taken into account increases, the favorability decreases.

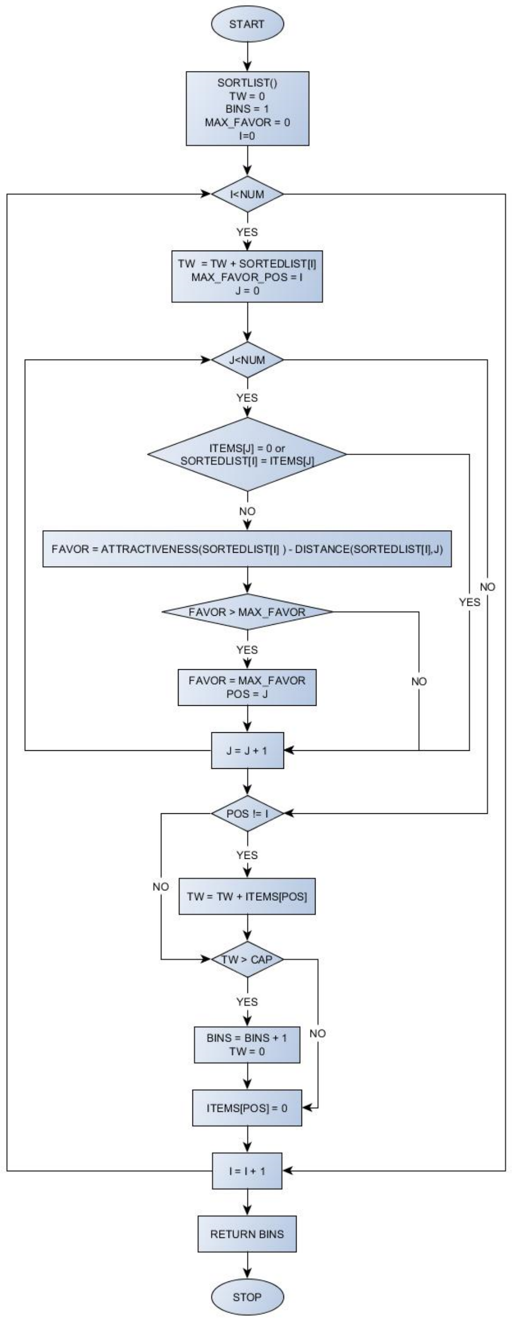
Algorithm:

implementFirefly()
```
sortList()
tw ← 0
bins ← 1
max_favor ← 0
for I ← 0 upto num do
    tw ← tw + sortedlist[i]
    max_favor_pos←i
    for j ← 0 upto num do
        if items[j]=0 or
        sortedlist[i]=items[j] do
            continue
        favor ←
        attractiveness(sortedlist[i]) –
        distance(sortedlist[i],j)
        if favor > max_favor do
            max_favor ← favor
            max_favor_pos ← j
    if max_favor_pos != i do
        tw ← tw + items[max_favor_pos]
    if tw>cap do
        bins ← bins+1
        tw ← 0
    items[max_favor_pos]=0
return bins
```

attractiveness(x)
```
return cap - x
```

distance(a,b)
```
for i ← 0 upto num do
    if items[i] = a do
        break
return |i-b|
```

Flow Diagram:



VII. References

Nature Inspired Algorithms: Success And Challenges – Xin-She Yang

Bachelor Thesis Implementation Of A Discrete Firefly Algorithm For The QAP Problem Within The Seage Framework – Karel Durkota – May 2011