Project Report

Aryan Singh 2018320
Aminesh Kumar Rai 2018314
Prakash Prajapt 2018350

Solving Bin Packing Problem Using Ant Colony Algorithm

Under the Guidance of Dr. Kusum Kumari Bharti

December 13, 2020

ABSTRACT

The aim of this project is to review different nature-inspired approaches to the bin packing problem. Then this project will outline the ant colony optimisation algorithm and apply it to the bin packing problem. The results from the experiments will be used in order to draw conclusions on how the parameters affect the algorithm and results.

1. INTRODUCTION

The bin packing problem is defined as a finite set of values O which represent the item sizes and two constants C and N where C is the bins capacity and N is the number of bins. The problem asks: is it possible to pack all the items into N bins. We can further define the specifics of the problem; is there an arrangement of O into N or fewer subsets, such that the sum of elements in any subset does not exceed C. BPP has many important applications such as multiprocessor scheduling, resource allocation, transportation planning, real-world planning, and packing and scheduling optimization problems. Many algorithms are used to deal with BPP, such as improved approximation algorithms, particle swarm optimization, and ant colony optimization, genetic approach.

2. Different Algorithm That Can Be Used To Solve the Bin Packing Problem

Here this project looks at a few different algorithms that have been inspired by nature. Knowing how the algorithms work generally is useful but in this section, this project will also focus on how the algorithms adapt to represent the bin packing problem.

2.1 First Fit Decreasing (FFD)

First fit decreasing (FFD) involves ordering the items in descending order and fitting them in the bins in chronological order, if the item doesn't fit in the current bin it tries the next bin. FFD yields very good results and is not a nature-inspired algorithm. Therefore FFD is generally accepted as the benchmark for testing how good algorithms are at bin packing. Another often used fast heuristic is best fit decreasing (BFD). The only difference from FFD is that the items are not placed in the first bin that can hold them, but in the best-filled bin that can hold them. This makes the algorithm slightly more complicated, but surprisingly enough, no better. Both heuristics have a guaranteed worst case performance of **11/9 Opt + 4**, in which Opt is the number of bins in the optimal solution to the problem [14].

2.2 Genetic Algorithm (HGGA)

Falkenauer's HGGA uses a grouping approach to solve the.BPP: the genetic algorithm (GA) works with whole bins rather than with individual items. Crossover essentially consists of choosing a selection of bins from the first parent and forcing the second parent to adopt these bins. Any bins in the second parent which conflict with the adopted bins have their items displaced, and a simple local search is used to replace them into the solution: free items are swapped with non-free items to make fuller bins which contain few large items rather than many small items. Any remaining free items are re-inserted into new bins using the FFD method. Mutation works in a similar way: a few random bins have their items displaced and then re-inserted via the local search procedure.

The genetic algorithm, when compared to the first-fit decreasing algorithm, is superior and can outperform it in what is described as 'tough conditions'.

2.3 Martello and Toth's reduction procedure

The basis of the MTP is the notion of dominating bins: when you have two bins B1 and B2, and there is a subset

{i1, . . . , il} of B1 and a partition {P1, . . . , Pl} of B2, so that for each item ij , there is a smaller or equal corresponding partition Pj , then B1 is said to dominate B2. This means that a solution which contains B1 will not have more bins than a solution containing B2. The MTP tries to find bins that dominate all other bins. When such a bin is found, the problem is reduced by removing the items of the dominating bin. In order to avoid that the algorithm runs into exponential time, only dominating bins of maximum three items are considered.

2.4 Ant Colony Optimisation

The first Ant Colony Optimisation algorithm was applied to a travelling salesperson problem in 1992 by Dorigo. The original algorithm has since been improved and applied to different problems.

Ant colony optimisation algorithms were inspired by the ant's ability to find the shortest path between their nest and a food source. They can do this by leaving a trail of pheromones wherever they walk. Other ants can smell the pheromone and follow it. When an ant has an option initially of two paths is chooses randomly between the two. The ants which run on the shortest path will go to the food and back to the nest faster. After they return to the nest there will be more pheromone on the shortest path. The initial random choice of two paths is then biased towards the path with more pheromones influencing more ants to take that path. After some time the entire colony will take the shortest path.

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)].[\eta(i,j)]^\beta}{\sum_{g \in J_k(i)}[\tau(i,g)].[\eta(i,g)]^\beta} & \text{if } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

In this equation, τ (i, j) is the pheromone between i and j and η(i, j) is a simple heuristic guiding the ant. The value of the heuristic is the inverse of the cost of the connection between i and j. So the preference of ant k in city i for city j is partly defined by the pheromone between i and j, and partly by the heuristic favourability of j after i. It is the parameter β which defines the relative importance of the heuristic information as opposed to the pheromone information. Jk(i) is the set of cities that have not yet been visited by ant k in city i. Once all ants have built a tour, the pheromone is updated. This is done according to these equations:

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)].[\eta(i,j)]^\beta}{\sum_{g \in J_k(i)}[\tau(i,g)].[\eta(i,g)]^\beta} & \text{if } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$\tau(i,j) = \rho.\tau(i,j) + \sum_{k=1}^{m} \Delta\tau_k(i,j) \quad (4)$$

$$\Delta\tau_k(i,j) = \begin{cases} \frac{1}{L_k} & \text{if } (i,j) \in \text{ tour of ant } k \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Equation (4) consists of two parts. The left part makes the pheromone on all edges decay. The speed of this decay is defined by ρ, the evaporation parameter. The right part increases the pheromone on all the edges visited by ants.

3.1 BPP Model Strategy Based On Ant Colony Optimization

The ant colony optimization (ACO) is a kind of simulant evolution algorithm based on real ant colony which is proposed by Italy experts M.Dorigo, V.Maniezzo and A. Colony.

The first step is to create an initial construction graph of random pheromones. The pheromones, in this case, are random values between 0 and 1.

The second step involves generating a set of paths. Each ant traversing the graph is one path and so in the algorithm, a list of bin numbers is used to represent a path. for example: [2,3,2,4,5] means the first item is put in bin 2, the second item is put in bin 3 and so on.

Then once the set of paths have been made the pheromone table must be updated. For every path a fitness is calculated, this is the difference in weights of the biggest bin and the smallest bin. In this algorithm to update the table, for each path add 100/fitness to the corresponding nodes in the pheromone table.

Next evaporation takes place, this involves multiplying every pheromone by a value between 0 and 1 specified as the evaporation rate.

The stopping criteria for this algorithm is 10000 fitness evaluations. Until that number is met, return to step 2. The number of fitness evaluations made each time depends upon how many ants you choose to use. With just 10 paths this will loop 1000 times before stopping.

The solution returned at the end may not be the best solution found in the entire 10000 evaluations but its the best solution found out of the number of ants you specify in that loop.

3.2 Results

For Bin Packing Problem 1 (BPP1) there were 4 experiments to attempt using the ant colony optimisation. For each experiment, the number of bins remained the same at 10, there were 500 items of weights 1 to 500. The algorithm ran until the number of fitness evaluations was equal to 10000. The number of ants traversing paths; P, and the evaporation rate is different for each experiment:

Experiment 1: P is set at 100, Evaporation Rate set at 0.9.

Experiment 2: P is set at 100, Evaporation Rate set at 0.6

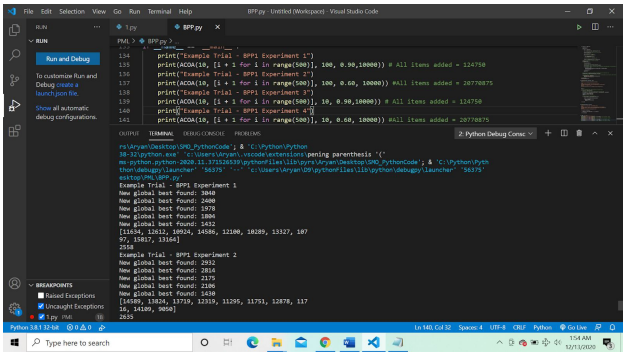
Experiment 3: P is set at 10, Evaporation Rate set at 0.9

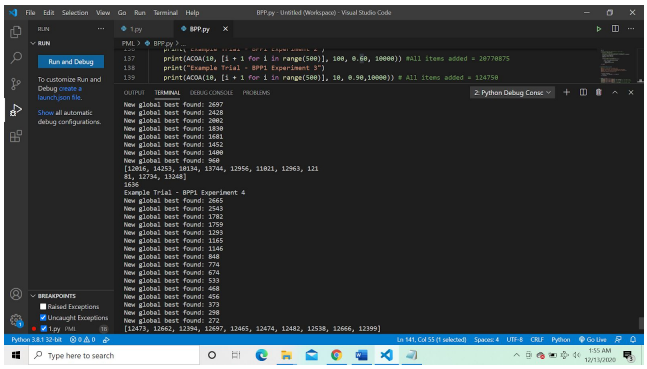
Experiment 4: P is set at 10, Evaporation Rate set at 0.6

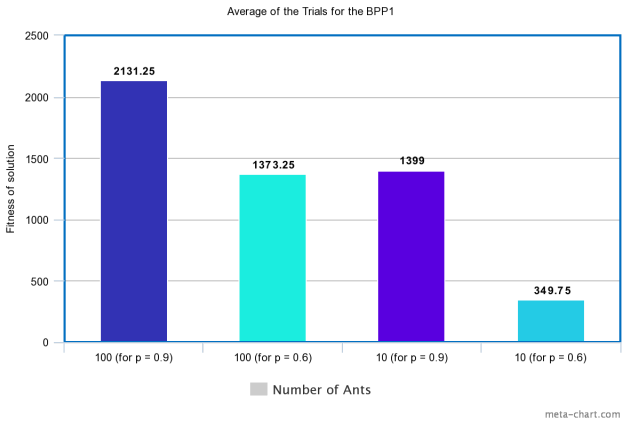
BPP1	Trial1	Trial2	Trial 3	Trial4	Average
Experiment1	2400	1978	1804	1432	2131.25
Experiment2	2814	2175	2106	1430	1373.25
Experiment3	1681	1452	1400	960	1399
Experiment4	456	373	298	272	349.75

The details are shown in the screenshot below:

PS.: Last 4 trials are taken into account.







4.1 Contribution Involved

Many nature inspired algorithm can be used to solve the Bin packing problem, but our approach was mainly focused on how to use the Ant Colony System algorithm in an efficient way to solve the problem.

So our first approach was to find the combination of parameters that produces the best result when certain constraints are specified like the weights of the items and the number of bins.

It is clear from both problems that the best parameters will be different depending upon the weight of the items.

For BPP1 we see a clear trend of 0.6 being the better evaporation rate compared to 0.9. We can see that when the number of ants remains the same the best solutions come from the experiments with 0.6 as the evaporation rate. The number of ants also affects the results, out of 100 ants vs 10 ants we see that the 10 ants yield a solution with a lower and therefore better fitness. Thus we can conclude for the BPP1 the parameters for the best results is P=10 where p is the number of ants, and evaporation rate set to 0.6.

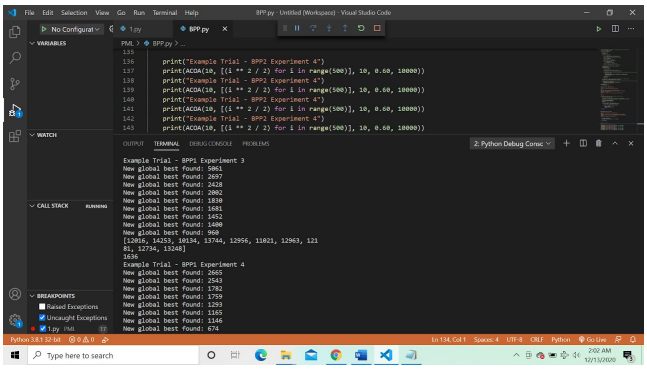
For BPP2 we see a similar trend to BPP1. The 10 ants outperformed the 100 ants with exception to experiment 4. 0.6 evaporation rate outperformed 0.9 with the exception again to experiment 4. For this reason, the best parameters in this problem are 10 ants with an evaporation rate of 0.9

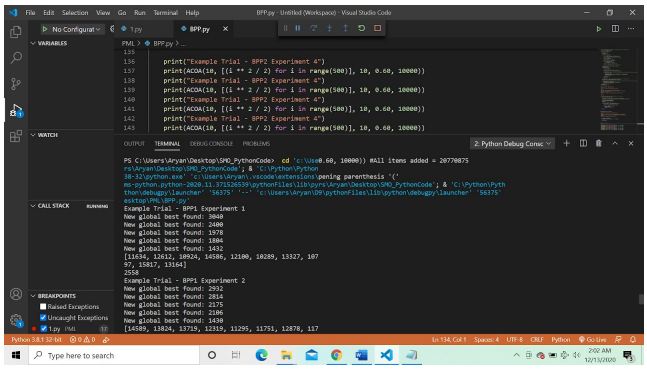
Having a more drastic evaporation rate can be beneficial, work and eliminate certain paths that are less efficient, however with the big data set like in BPP2 the 0.6 converges too fast and so the majority of the time it will get bad results. It is beneficial for big data sets to have a less drastic evaporation rate like 0.9. The results in the experiment reflect this.

The number of ants does not affect the time it takes to run the entire algorithm because the algorithm used is based on fitness evaluations, and so no matter how many ants are used the same number of evaluations are done. The ants do however affect the performance in terms of solutions, from the experiment we see that the fewer ants used are yield a better solution because the fitness evaluations are fixed for all experiments.

The bins used affect the timing of the algorithm, the more bins the longer it takes to run, this is because there are more probabilities to calculate for each ant path. The number of fitness evaluations also affects the timing, the more fitness evaluations the longer the algorithm takes to run. With more fitness evaluations the solution improves as a general trend, this occurs until a local optimum has been reached and the solution will stay the same.

Changing the evaporation rate although would not affect the speed of the algorithm, it will get differing results and so it is possible to find an optimum rate for different items. To improve the performance in cases of a big data set for items a less drastic evaporation rate gets a better solution (for example 0.9 is less drastic than 0.6)





The above pictures are the BPP2 experiments.

4.1 Comparative Analysis And Comparison with Different Algorithm

For the ant colony optimisation algorithm that was used in the experiments, the main limiting factor to getting better results was the limit of 10000 fitness evaluations. Increasing this parameter will gain better results

In the literature review, the algorithms were compared to first fit decreasing algorithms, however, in this case, it cannot be done. This is because the bins have no maximum capacity so all the items will be put in one bin for FFD.

The ant colony optimisation problem only shows the best solution of the current ant paths.

The genetic algorithm given enough generations will converge onto a local optimum. Considering in the experiments the fitness evaluations were limited to 10000, we think the genetic algorithm will outperform it based solely on more time to converge the solution. For experiment 4 of BPP2, we think that the genetic algorithm will perform better. The mutations tend to be small and so it won't converge too fast, unlike the ant colony optimisation example.

Here is the time taken by different algorithm to solve the bin packing problem.

Prob	HGGA		MTP		ACO	
	bins	time	bins	time	bins	time
u120	+2	381	+2	370	+2	376
u250	+3	1337	+12	1516	+12	1414
u500	0	1015	+44	1535	+42	1487
u1000	0	7059	+78	9393	+70	9272

Table 2: Pure ACO Results for Uniform Bin Packing Problems

4.3 Conclusions

A few nature-inspired approaches to the bin packing problem have been presented. Genetic algorithm, Cuckoo Search and Ant Colony Optimisation. The ant colony approach was implemented, and the parameters of the algorithm tested to see how they affect the result. Although it's not the most efficient nature-inspired algorithm, it was useful to discover how depending upon the items of the problem, different parameters give better solutions.

5. REFERENCES

1. Study on Improved Ant Colony Optimization on Bin Packing Problem
   https://ieeexplore.ieee.org/document/5540875
2. Ant Colony Optimisation and Local Search for Bin Packing
   http://people.idsia.ch/~frederick/levine-jors03.pdf