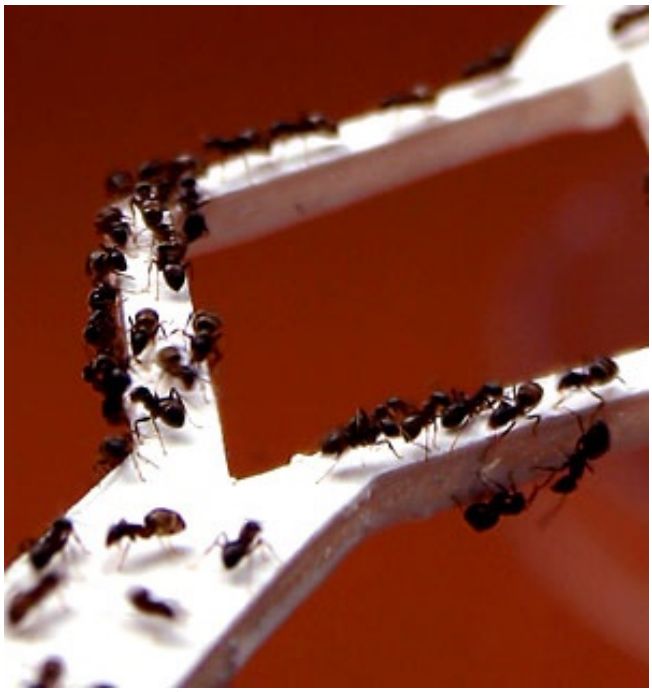

source: www.quora.com/What-are-some-modern-day-uses-of-Artificial-Distributed-Intelligence

BIN-PACKING WITH ANT COLONY OPTIMISATION

ECM3412 – Nature-Inspired Computation,
Assignment One

ABSTRACT

This paper first discusses the other nature-inspired algorithms applied to the bin packing problem. An experiment on the effectiveness of ant colony optimisation for bin packing is described and implemented in a python program. Results have shown that a higher evaporation rate finds solutions at a faster rate although at the cost of quality. A higher frequency of pheromone updates yields better solutions but requires more processing power. The genetic grouping algorithm discussed in the literature is a worthy opponent of ant colony optimisation, likely providing better results due to its superior adaptation to the problem.

Schiedermair, Leon
University of Exeter

1. Introduction

Ant colony optimisation (ACO), a member of the swarm intelligence family, is a probabilistic procedure used to solve computational problems by finding good paths across graphs. Originally suggested by Marco Dorigo in 1991, the first version of the algorithm was tasked with searching for an optimal route in a graph, mimicking the behaviour of ants finding a path between a food source and their colony (Colorni, Dorigo and Maniezzo, 1991). The initial concept has since been diversified, providing solutions to a broader range of numerical problems, drawing even more attention to the numerous characteristics of the behaviour of ants.

The Bin Packing Problem (BPP) has many variants, but focusses mainly on fitting objects of different sizes into a finite number of bins. The goal is often to minimise the number of bins used. Some variants of BPP include 2D packing, packing by weight, packing by cost, linear packing, and so on. Along with these variants are many real-world applications, such as generating file backups in media, loading vehicles with weight or capacity restraints and filling up shipment containers.

This paper discusses previous attempts at using nature-inspired algorithms to solve the BPP, providing a critical analysis of their effectiveness. An implementation of ant colony optimisation to solve the BPP is explored, using a Python program to facilitate the investigation into its suitability. The goal is to find out which ACO parameter values yield the best results for bin packing.

2. Literature Review

2.1. A Genetic Algorithm for Bin Packing

A Hybrid Grouping Genetic Algorithm for Bin Packing (Falkenauer, 1996) first describes the genetic grouping paradigm and compares it to the definitive Holland-style genetic algorithm (Holland, 1992), as well as the ordering genetic algorithm. It is then shown how the Bin Packing genetic grouping algorithm may be improved using local optimisation inspired by a dominance criterion. The bin packing problem described in this paper involves a finite set O of numbers corresponding to the sizes of the items to be accumulated in the bins. A bin's capacity is denoted by the constant C, and N is the total number of bins. The goal here is to determine whether it is possible to 'fit' all items into N number of bins; in other words, to find if there exists a partition of O in N or fewer subsets, where the sum of all elements within any of the subsets does not exceed C. The main focus of this paper is to find which is the *best* way of fitting items into bins - that is, to find the least number of subsets within the partition mentioned above.

Another aspect of this problem concentrates on grouping members of a set U to at least one other group of items, where every item is placed in precisely one group – this means trying to find *groupings* of the items. The aim of grouping is to find an optimisation of a cost function for a set of all acceptable groupings.

2.1.1. Standard Crossover Method Inadequacy

Crossover is a type of genetic operator used for creating variations in the structure of a chromosome from generation to generation. It is comparable to reproduction or biological crossover, upon which the fundamentals of all genetic algorithms are based.

When applying a typical 2-point crossover to the chromosomes `A|BC|ADD` and `C|AD|CBB` where '|' signifies a crossover point, the resulting children would be `CBCCBB` and `AADADD`. Without mutation, a combination of two indistinguishable chromosomes must generate progeny that is identical to its parents. On the other hand, the children resulting from the two chromosomes above, encode 'solutions' that have nothing in common with their parents' encoded solution. There are only two groups: `C` and `B` in the first child, `A` and `D` in the second, compared to the four present in the parent chromosomes. Although the schemata are transmitted well with regard to the chromosomes when using this standard model of crossover/encoding, their significance regarding the problem at hand – optimising a cost function – becomes lost during the recombination process.

2.1.2. Standard Mutation Method Inadequacy

Another genetic operator, mutation, is used to provide genetic diversity between direct descendants of chromosomes. Analogous to biological mutation – it works by altering at least one gene (or value) within a chromosome from its original state. When applying the standard mutation encoding to the chromosome `ABDBAC`, a standard mutation of the chromosome might produce `ABDEAC`.

Although this may be advantageous, as the allele `E`, which may be missing from the population, is present within the genetic pool. The problems arise when the algorithm begins to reach a good solution, where large clusters of matching alleles are emerging. When using standard mutation, the chromosome `AAABBB` might produce, for instance, `AACBBB` which holds a 'group' consisting of only one element `C`. As the grouping of elements typically accounts for an increase in fitness, this mutated chromosome will presumably show an abrupt decline in fitness when compared to other non-mutated chromosomes. Therefore, classic mutation has proved to be rather detrimental once the genetic algorithm begins to approach a good solution for the grouping problem.

2.1.3. The Grouping Genetic Algorithm

As it has been shown, the standard genetic operators are unsuitable for grouping problems. This is due to the structure of these basic chromosomes being *item oriented*, rather than *group oriented*. Consequently, this method of encoding is not well adapted to the particular cost function that is to be optimised.

The Grouping Genetic Algorithm is different from the standard genetic algorithm in two significant ways. Firstly, a distinctive encoding scheme is utilised to ensure that the appropriate structures of grouping problems correspond to genes within chromosomes. Secondly, with the encoding provided, certain genetic operators are applied that are appropriate for the chromosomes.

As a solution to the above mentioned encoding problems, a new encoding scheme has been used. Consider a similar chromosome to the previous examples, where the items to be packed have been numbered from 0 to 5. The *item* part of the chromosome may explicitly be written as

`012345`
`ADBCEB:…`,

signifying that the item `0` has been placed in bin `A`, item `1` is in `D`, `2` and `5` are both in `B`, and so on. The *group* part, expressing only the groups is `…:BECDA`.

It is important to recognise that genetic operators are fully compatible with the group part of the chromosomes, where the item portion of a chromosome simply serves as a way to categorise the items into groups. This encoding scheme allows the alleles to symbolise the groups. The motivation behind this is due to the nature of grouping problems, where the groups are the important building blocks – the minutest

part of a solution that may express any expectation of quality for the solution they belong to. Therefore, this allows them to act as an estimator of quality for the areas of search space they inhabit.

2.1.4. Grouping Genetic Algorithm Crossover

The following defines the crossover algorithm used for the genetic grouping algorithm, as described in *A Hybrid Grouping Genetic Algorithm for Bin Packing* (Falkenauer, 1996):

1. Select at random two crossing sites, delimiting the *crossing section*, in each of the two parents.
2. *Inject* the contents of the crossing section of the first parent at the first crossing site of the second parent. Recall that the crossover works with the group part of the chromosome, so this means injecting some of the *groups* from the first parent into the second.
3. Eliminate all *items* now occurring twice from the groups they were members of in the second parent, so that the 'old' membership of these items gives way to the membership specified by the 'new' injected groups. Consequently, *some* of the 'old' groups coming from the second parent are altered: they do not contain all the same items anymore, since some of those items had to be eliminated.
4. If necessary, *adapt* the resulting groups, according to the hard constraints and the cost function to optimize. At this stage, local problem-dependent heuristics can be applied.
5. Apply the points 2. through 4. to the two parents with their roles permuted in order to generate the second child.

This shows that the idea behind the grouping genetic algorithm crossover is to promote promising *groups* by inheritance. Mutation will be ignored, as a mutation operator for grouping problems must work with groups rather than items.

2.1.5. Experiment Setup and Results

In order to assess the merit of the algorithm, the grouping genetic algorithm (HGGA) was compared to the MTP procedure of (Martello and Toth, 1990). MTP was chosen as a benchmark as it is considered by many to be one of the best methods for the Bin Packing Problem to date. The experiments discussed in this summary involve a population of 100 individuals and a tournament of size 2 as the selection strategy.

Upwards of 250 items, the exponential characteristics of MTP begin to show, and the results show the superiority of the GGA in two respects: the GGA provides better solutions and does so faster than MTP. For seventeen out of the twenty instances of 250 items, the GGA has discovered a solution using the least number of bins – the global optimum. For the remaining three runs, the GGA's solution had the same number of bins as the MTP's, so it is assumed that these were also globally optimal. The results on instances of 500 items and 1000 items further confirm the superiority of the GGA in comparison to the MTP procedure.

2.2. Simulated Annealing for Bin Packing

Simulated Annealing (SA) is used to solve optimisation problems that are unconstrained or bound-constrained. The method is based on the physical process of heating a material, then gradually decreasing the heat to reduce defects, therefore minimising the system energy.

Simulated Annealing Based Algorithm for the 2D Bin Packing Problem with Impurities (Beisiegel *et al.*, 2006) considers a more complicated real-world problem originating in the steel industry. The bins modelled in this paper are inhomogeneous sheets, each with impurities, where it is assumed that an impure area is rectangular. Each bin contains a set of impurities, size, as well as the location of every impurity in the bin. This leads to a finite number of bins as they are no longer identical. Moreover, this introduces a linear order on the set of bins; the first bin must be used first, and if additional bins are required, the second bin is used, then the third, and so on. The goal here is to find solutions with a minimal number of bins.

2.2.1. Tailored Simulated Annealing

To solve this NP-hard problem, a tailored Simulated Annealing algorithm was developed. The starting point for this algorithm was one of the greedy type, it follows a polynomial time heuristic meaning simulated annealing can be started with a low temperature. There are already many encoding schemes built for 2D rectangular bin packing problems, however the method used in this paper is an oriented tree representation[1].

Within the oriented root tree, each node corresponds to an item. This gives $O(n^2)$ neighbours, where n is the total number of nodes. The simulated annealing algorithm is then applied to each bin in parallel. The initial set of trees may be selected at random and the objective function $F(T)$, which is to be minimised, is equal to the used part of the bin.

2.2.2. Results

The tailored simulated annealing algorithm was tested on real world, and randomly generated instances. The bin impurities were generated at random for every bin, and the impurity size is also randomised and represented as items. For the real-world tests, the instances had a relatively small dimension of $n \leq 30$, and the algorithm found optimal solutions for all of them. However, for $n = 100$ the lower bound of the test results is quite far from the optimal solution and errors are relatively large. Therefore, for instances of a larger scale the heuristic solutions are fairly close to the lower bound and relative errors are small.

3. Bin Packing with Ant Colony Optimisation

Written in Python, a program is used to test the effectiveness of ant colony optimisation for bin packing. The results of this experiment will be used to compare how altering some of the algorithm parameters may produce more optimal results for variations of the bin packing problem.

The fitness of a solution is given by calculating the difference d between the heaviest and lightest bins, where the goal is to minimise this difference. There are two specific bin packing problems to be addressed, both of which involve 500 items to be packed into a number of b bins. The first (BPP1), uses $b = 10$ and the weight of an item i, (where i starts at 1 and finishes at 500) is $2i$. In the second (BPP2), $b = 50$ and the weight of item i is i^2. A single trial consists of 10,000 fitness evaluations - or iterations - of the ant colony optimisation algorithm. The process terminates upon completion of the last fitness evaluation.

Five trials of each of the following instances of both problems are run:

- $p = 100, e = 0.9$

[1] Also known as a Polytree, it is a directed, acyclic graph whose underlying undirected graph is a tree.

- $p = 100, e = 0.6$
- $p = 10, e = 0.9$
- $p = 10, e = 0.6$

3.1. Description of Results

After running a total of forty trials, the results have shown that in general, the parameter $p = 10$ (orange and yellow) provides the best solutions for BPP1 with 10,000 fitness evaluations.

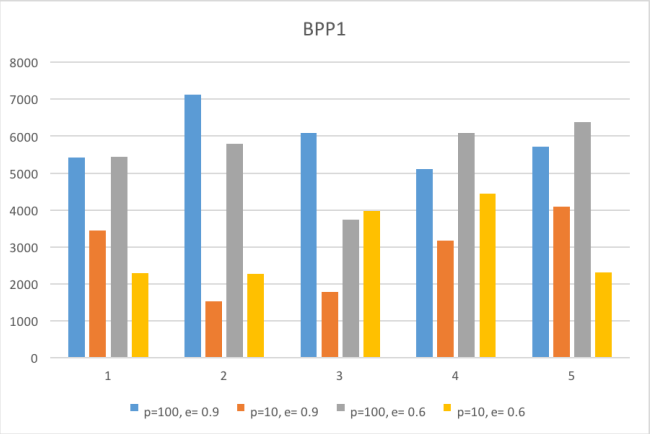

Figure 1

Figure 1 shows that the best fitness results for $p = 100$ (blue and grey) are worse for BPP1 across all trials, compared to $p = 10$. However, in Figure 2, there is no clear winner as the results are fairly spread out across all runs.

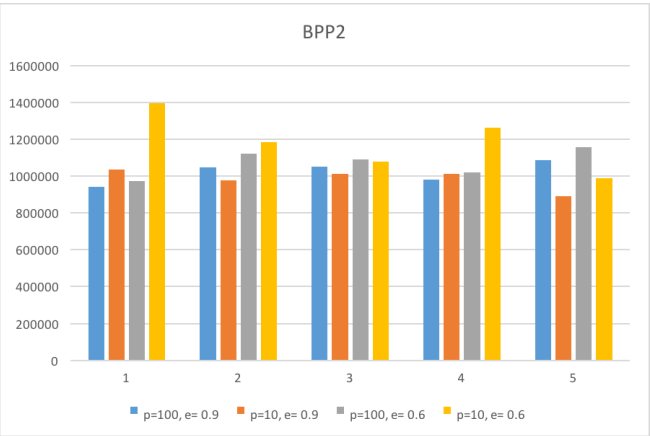

Figure 2

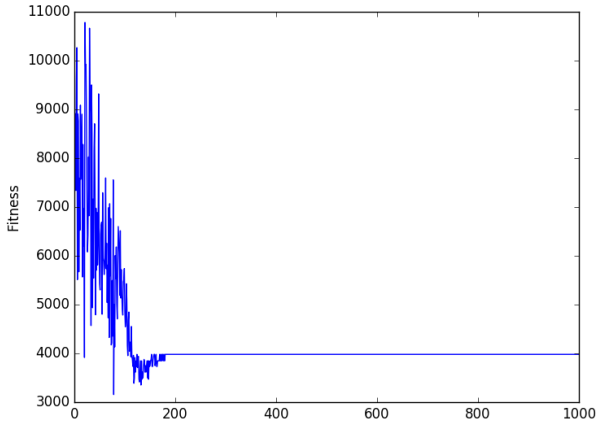

Figure 3: BPP1 p=10, e=0.6

Figure 3 shows how a more aggressive evaporation rate causes the graph to converge to a single value more quickly, although it is not an optimal solution as there are fitness evaluations lower than the final result earlier on in the graph.

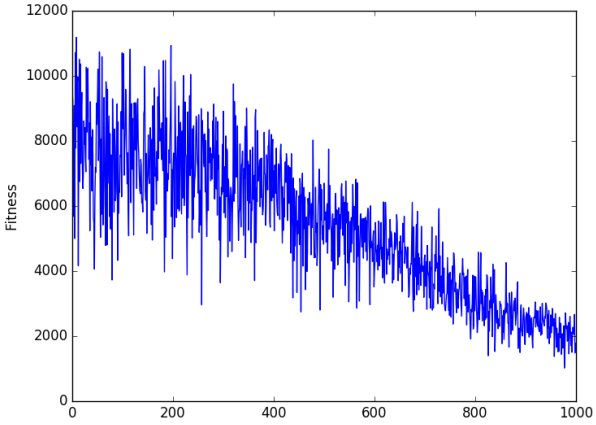

Figure 4: BPP1 p=10, e=0.9

With an evaporation rate of 0.9, figure 4 shows the graph is slowly converging to a better solution than for $e = 0.6$, however it will require a larger number of runs.

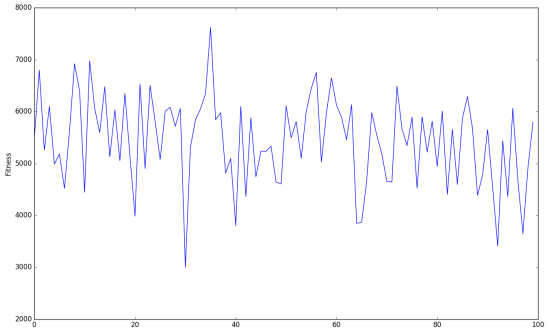

Figure 5: BPP1 p=100, e=0.6

Figures 5 shows a very similar result to that of $p = 100$, $e = 0.9$, where the best fitness evaluation of p number of generated paths has been plotted. However, if left for a much larger number of fitness evaluations, it eventually converges to a near optimal solution, where the best of five trials gave a fitness result of 188, as shown in figure 6.

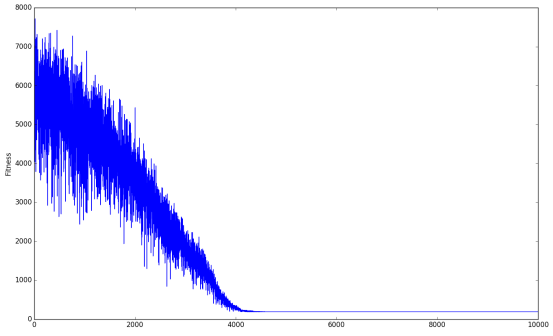

Figure 6: BPP1 p=100, e=0.9, 1,000,000 fitness evaluations

Figure 7 shows a fast converging of fitness results, although it is not an optimal solution and appears to be quite sporadic.

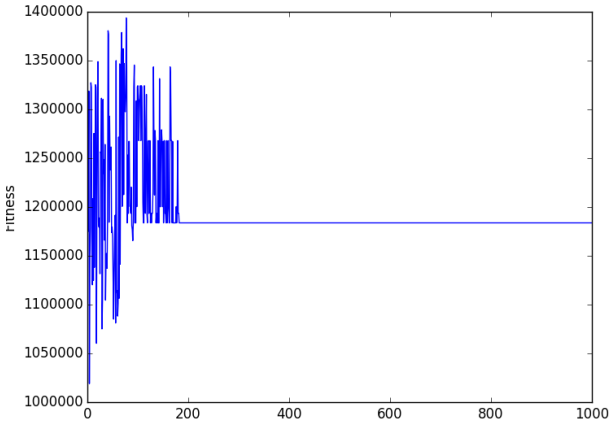

Figure 7: BPP2 p=10, e=0.6

Problem	Trial	p	e	best fitness
1	1	100	0.9	5420
1	2	100	0.9	7130
1	3	100	0.9	6078
1	4	100	0.9	5116
1	5	100	0.9	5720
1	1	100	0.6	5450
1	2	100	0.6	5796
1	3	100	0.6	3744
1	4	100	0.6	6094
1	5	100	0.6	6376
1	1	10	0.9	3456
1	2	10	0.9	1528
1	3	10	0.9	1782
1	4	10	0.9	3166
1	5	10	0.9	4100
1	1	10	0.6	2288
1	2	10	0.6	2276
1	3	10	0.6	3980
1	4	10	0.6	4452
1	5	10	0.6	2322
2	1	100	0.9	942418
2	2	100	0.9	1045788
2	3	100	0.9	1050379
2	4	100	0.9	982255
2	5	100	0.9	1086888
2	1	100	0.6	972801
2	2	100	0.6	1119641
2	3	100	0.6	1090069
2	4	100	0.6	1020156
2	5	100	0.6	1155872
2	1	10	0.9	1036623
2	2	10	0.9	978419
2	3	10	0.9	1011920
2	4	10	0.9	1011973
2	5	10	0.9	889337
2	1	10	0.6	1395225
2	2	10	0.6	1183741
2	3	10	0.6	1077581
2	4	10	0.6	1260924
2	5	10	0.6	990104

Figure 8: All results

3.2. Discussion and Further Work

It is clear from the results that the parameters $p = 10$ and $e = 0.9$ provide the best solutions to the given bin packing problems. The average results of the trials proved superior to the other combinations of parameters as well as the best single trial. With a smaller value of p, pheromone trails are updated more often, so better paths are more likely to be chosen. One variation of the algorithm could involve updating pheromone values immediately after an ant finds a path which may provide better results, although this would mean relying on a smaller number of ants to find good solutions; an issue to look into in future work. The difference in results between $p = 10$ and p=100 is quite large, this is due to the fact that a high portion of the results are dependent on the pheromone trails. A p value of 100 will not be adequate to cause convergence to a single fitness with only 10,000 fitness evaluations, and figure 6 shows that with more time, it could find a very good solution.

The second best set of results came from e=0.6 and p=10, where fitness improves at a much faster rate but converges to a suboptimal solution. This is due to the more aggressive evaporation rate causing the less explored paths to become increasingly unlikely to be chosen, until they 'disappear' altogether.

These results allow us to conclude that lower p values contribute towards frequent pheromone updates and in turn allows the algorithm to assess the quality of paths more often. However, this significantly increased computation time compared to $p = 100$. A lower p value will therefore yield higher quality results but at the cost of computational efficiency.

Making adjustments to the evaporation rate had no effect on computational efficiency, although the difference in results are relatively large. Although not as drastic as the comparison of different p values, the graphs have shown a faster rate of convergence. This is due to less used paths evaporating at a faster rate, consequently increasing the relative impact of pheromone updates causing ants to stick to certain paths.

It is possible that the grouping genetic algorithm discussed in the literature review would provide better results for this particular bin packing problem, especially since it is tailored towards grouping problems of this type. However, results may prove similar to ant colony optimisation due to the random nature of the two. The ant colony optimisation algorithm has not been drastically adapted to the bin packing problem and has provided some inconsistent results, but this is also due to randomness.

4. References

Beisiegel, B., Kallrath, J., Kochetov, Y. and Rudnev, A. (2006) 'Simulated Annealing Based Algorithm for the 2D Bin Packing Problem with Impurities', *Operations Research Proceedings 2005 Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Bremen, September 7–9, 2005*, pp. 1–6.

Colorni, A., Dorigo, M. and Maniezzo, V. (1991) 'Distributed Optimization by Ant Colonies', *Proceedings of the first European Conference on Artificial Life*, 142(or D), pp. 134–142. doi: 10.1016/S0303-2647(97)01708-5.

Falkenauer, E. (1996) 'A hybrid grouping genetic algorithm for bin packing', *Journal of Heuristics*, 2(1), pp. 5–30. doi: 10.1007/BF00226291.

Holland, J. H. (1992) 'Genetic Algorithms - Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand', *Scientific American*, pp. 66–72.

Martello, S. and Toth, P. (1990) 'Lower bounds and reduction procedures for the bin packing problem', *Discrete Applied Mathematics*, 28(1), pp. 59–70. doi: 10.1016/0166-218X(90)90094-S.