Ant Colony Optimisation for Bin Packing and Cutting Stock Problems

Frederick Ducatelle
Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
fredduc@dai.ed.ac.uk

John Levine
Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
johnl@aiai.ed.ac.uk

Abstract

The Bin Packing and Cutting Stock Problems are well-known NP-hard combinatorial optimisation problems with many applications. A number of evolutionary computation techniques have been applied to these problems, including genetic algorithms and evolutionary strategies. In this work, we investigate the use of Dorigo's Ant Colony Optimisation meta-heuristic to solve Bin Packing and Cutting Stock Problems. We show that the technique works well and can outperform other EC techniques. It is also shown to be quite sensitive to the relative weighing of the heuristic (first fit decreasing) as opposed to pheromone trail information.

1 Introduction

The Bin Packing Problem (BPP) and the Cutting Stock Problem (CSP) are two classes of well-known NP-hard combinatorial optimisation problems (see [10, 14, 2]). The BPP is concerned with combining items into bins of a certain capacity so as to minimise the total number of bins, whereas the CSP is concerned with cutting items from stocks of a certain length, minimising the total number of stocks. Obviously the two problem classes are very much related, and our approach will be able to tackle both of them.

Traditional approaches for the CSP are heuristics or methods based on linear programming ([14]). BPP instances are usually solved using simple heuristics ([10]), or the Reduction Algorithm of Martello and Toth ([17]). Recently, several researchers have started to apply evolutionary algorithms to both problem classes ([11, 15, 16, 18, 21]). The most successful of these is Falkenauer's Hybrid Grouping Genetic Algorithm (HGGA) ([11]).

In this article, we propose an Ant Colony Optimisation (ACO) approach to the BPP and the CSP. The first ACO algorithm was developed by Dorigo as his PhD thesis in 1992, and published under the name Ant System (AS) in [8] in 1996. It was an application for the Travelling Salesman Problem (TSP), loosely based on the path-finding abilities of real ants. Combining the use of heuristic information and an artificial pheromone trail, which was reinforced by good solutions, AS was able to find optimal solutions for some smaller TSP instances. After the first publication, many researchers have proposed improvements to the original AS, and applied it successfully to a whole range of different problems (see [9] for an overview). No one has used it for the BPP or the CSP, however, apart from a hybrid approach by Bilchev, who uses ACO to combine genetic algorithms and a many-agent search model for the BPP (see [1]).

The rest of this article is organised as follows. Section 2 describes the BPP and the CSP. Section 3 introduces ACO algorithms. Section 4 contains a detailed explanation of how we applied ACO to the BPP and the CSP, and section 5 gives an overview of the experimental results. Section 6 concludes with a summary of the paper and a quick look at our work in progress on this subject.

2 Bin Packing and Cutting Stock Problems

In the traditional one-dimensional BPP, we are presented with a fixed assortment of items, each of a certain weight. The aim is to combine these items into bins of a fixed maximum weight, minimising the total number of bins used. In the traditional one-dimensional CSP, there is a fixed assortment of items of a certain length given. They have to be cut from stocks of a fixed length. The aim is to minimise the number of stocks used (a very common situation in the wood industry for example).

From the above description, it should be clear that these two problems are very similar. In fact, according to Dyckhoff ([10]), the only difference between them lies in the assortment of items: in the BPP there are traditionally many items of many different lengths, whereas in the CSP, the items are usually only of a few different lengths (so there are many items of the same length). This means that the difference between the CSP and the BPP is a rather subjective and gradual one.

Even though it is hard to draw a line between the two problem classes, the difference between them dictates different solution approaches. Because the CSP has a lot of items of the same length, stocks are often cut down by the same pattern. Therefore it makes sense to first construct good patterns, and then decide how many stocks will be cut according to each pattern. Traditional solution methods follow this approach. A distinction can be made between linear programming based approaches and sequential heuristic procedures (see [14]). For the BPP, this approach does not make sense, as each pattern would most likely only be used once. One of the best approaches for the BPP is Martello and Toth's Reduction Algorithm ([17]). Apart from this, simple fast heuristics like First Fit Decreasing or Best Fit Decreasing are most often used.

Recently, new solution methods for both problem classes have been proposed, based on evolutionary strategies. People have tried out various forms of order-based and grouping-based genetic algorithms ([15, 21]), hybrid methods ([1, 11, 18]), and also evolutionary programming ([16]). The most successful of these approaches is Falkenauer's HGGA ([11]), which combines a grouping based genetic algorithm with elements of Martello and Toth's Reduction Algorithm.

3 Ant Colony Optimisation

ACO is a multi-agent meta-heuristic for combinatorial optimisation and other problems. It is inspired by the capability of real ants to find the shortest path between their nest and a food source. The first ACO algorithm, AS, was an application to solve the TSP, developed in 1992 by Dorigo. AS became very popular after its publication in 1996 (see [8]). Many researchers have since developed improvements to the original algorithm, and have applied them to a range of different problems.

ACO algorithms were originally inspired by the ability of real ants to find the shortest path between their nest and a food source. The key to this ability lies in the fact that ants leave a pheromone trail behind while walking (see [3]). Other ants can smell this pheromone, and follow it. When a colony of ants is presented with two possible paths, each ant initially chooses one randomly, resulting in 50% going over each path. It is clear, however, that the ants using the shortest path will be back faster. So, immediately after their return there will be more pheromone on the shortest path, influencing other ants to follow this path. After some time, this results in the whole colony following the shortest path.

AS is a constructive meta-heuristic for the TSP[1], loosely based on this biological metaphor. It associates an amount of pheromone $\tau(i,j)$ with the connection between two cities i and j. Each ant is placed on a random start city, and builds a solution going from city to city, until it has visited all of them. The probability that an ant k in a city i chooses to go to a city j next is given by equation 1:

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)].[\eta(i,j)]^\beta}{\sum_{g \in J_k(i)} [\tau(i,g)].[\eta(i,g)]^\beta} & \text{if } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In this equation, $\tau(i,j)$ is the pheromone between i and j and $\eta(i,j)$ 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. $J_k(i)$ 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:

$$\tau(i,j) = \rho.\tau(i,j) + \sum_{k=1}^{m} \Delta\tau_k(i,j) \tag{2}$$

$$\Delta\tau_k(i,j) = \begin{cases} \frac{1}{L_k} & \text{if } (i,j) \in \text{ tour of ant } k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Equation (2) consists of two parts. The left part makes the pheromone on all edges decay. The speed of this

[1] An instance of the TSP contains a number of cities and connection costs between them. The aim is to build a closed tour of all the cities while minimising the total cost of it.

decay is defined by ρ, the evaporation parameter. The right part increases the pheromone on all the edges visited by ants. The amount of pheromone an ant k deposits on an edge is defined by L_k, the length of the tour created by that ant. In this way, the increase of pheromone for an edge depends on the number of ants that use this edge, and on the quality of the solutions found by those ants.

AS performed well on relatively small instances of the TSP, but could not compete with other solution approaches on larger instances. Later, many improvements to the original AS have been proposed, which also performed well on the bigger problems. Examples of these improvements are Ant Colony System ([7]) and MAX-MIN Ant System ([20]). Also, these algorithms have been applied successfully to other problems, such as the Quadratic Assignment Problem ([13]), the flow shop problem ([19]) and Network Routing ([4]). A good overview can be found in [3] or [9].

4 The Approach

In this section, we describe how we adapted the ACO algorithm to solve the BPP and the CSP. We will talk about the pheromone trail definition, the heuristic, the building of a solution by the ants, the fitness function and the updating of the pheromone trail.

4.1 The Pheromone Trail Definition

The quality of an ACO application depends very much on the definition of the meaning of the pheromone trail ([9]). It is crucial to choose a definition conform the nature of the problem. The BPP and the CSP are grouping problems. What you essentially want to do, is split the items into groups. This is in contrast to the TSP and most other problems ACO has been applied to. The TSP is an ordering problem: the aim is to put the different cities in a certain order. This is translated in the meaning of the pheromone trail: it encodes the favourability of visiting a certain city j after another city i.

To our knowledge, there is only one ACO application for a grouping problem. It is Costa and Hertz' AntCol ([6]), an ACO solution for the Graph Colouring Problem (GCP). In the GCP, a set of nodes is given, with undirected edges between them. The aim is to colour the nodes in such a way that no nodes of the same colour are connected. So, in fact, you want to group the nodes into colours. Costa and Hertz use a grouping based approach, in which the pheromone trail between node i and node j encodes the favourability of having these nodes in the same colour. The pheromone matrix is of course symmetric ($\tau(i,j) = \tau(j,i)$).

We will define our pheromone trail in the same way as Costa and Hertz: $\tau(i,j)$ encodes the favourability of having an item of length i and length j in the same bin (or stock). There is of course one important difference between the GCP on one side and the BPP and the CSP on the other: in the GCP, there is only one node i and one node j, whereas in the BPP, and even more so in the CSP, there are several items of length i and length j. We initially thought that this might give a problem, as it could for example be favourable to combine i and j in n bins, but not in the $n{+}1^{\text{th}}$ bin. We tried to solve this by making changes to the pheromone trail while it was being used by the ants (lower the pheromone between i and j every time i and j were combined in a bin or stock). In later tests, however, this turned out to be unnecessary: the algorithm worked better without the changes.

4.2 The Heuristic

Another important feature of an ACO implementation is the choice of a good heuristic, which will be used in combination with the pheromone information to build solutions. We chose to use one of the simple heuristics for solving the BPP: First Fit Decreasing (FFD). In FFD, the items are first sorted in order of non-increasing weight, and then each item is placed in the first bin it still fits in. This heuristic is less complicated than Best Fit Decreasing[2], but is guaranteed to perform equally well: a worst case performance of $\frac{11}{9}OPT + 4$, in which OPT is the number of bins in the optimal solution to the problem ([5]).

As our ACO approach is constructive (we will be filling the bins one by one, instead of placing the items one by one), we have to reformulate FFD slightly. We fill a bin with the biggest items left that still fit in it. If no items are light enough to fit in the bin, a new bin is started. This results in the FFD solution, but is more useful for us: the heuristic favourability of an item is now given by its weight (or size).

4.3 Building a Solution

The pheromone trail and the heuristic information defined above will now be used by the ants to build solutions. Every ant starts with the set of all items to

[2] In Best Fit Decreasing, the items are again sorted in order of non-increasing weight. Each item is then placed in the best filled bin it fits in.

be placed and an empty bin. It will add the items one by one to its bin, until none of the items left are light enough to fit in the bin. Then the bin is closed, and a new one is started. The probability that an ant k will choose an item j as the next item for its current bin b in the partial solution s is given by equation 4:

$$p_k(s,b,j) = \begin{cases} \frac{[\tau_b(j)].[\eta(j)]^\beta}{\sum_{g \in J_k(s,b)} [\tau_b(g)].[\eta(g)]^\beta} & \text{if } j \in J_k(s,b) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In this equation, $J_k(s,b)$ is the set of items that qualify for inclusion in the current bin. They are the items that are still left after partial solution s is formed, and are light enough to fit in bin b. $\eta(j)$ is the weight of item j. The pheromone value $\tau_b(j)$ for an item j in a bin b is given in equation 5 below. It is the sum of all the pheromone values between item j and the items i that are already in bin b, divided by the number of items in b. If b is empty, $\tau_b(j)$ is set to 1. This approach is similar to the one followed by Costa and Hertz.

$$\tau_b(j) = \begin{cases} \frac{\sum_{i \in b} \tau(i,j)}{|b|} & \text{if } b \neq \{\} \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

4.4 The Fitness Function

In order to guide the algorithm towards good solutions, we need to be able to assess the quality of the solutions. So we need a fitness function. A straightforward choice would be to take the inverse of the number of bins. As Falkenauer ([11]) points out, however, this results in a very unfriendly fitness landscape. Often there are many combinations possible with just one bin more than the optimal solution. If these all get the same fitness value, there is no way they can guide the algorithm towards an optimum, and the problem becomes a needle-in-a-haystack.

So, instead, we chose to use the function proposed by Falkenauer and Delchambre in [12] to define the fitness of a solution s:

$$f(s) = \frac{\sum_{i=1}^{N} (F_i/C)^k}{N} \tag{6}$$

In this equation is N the number of bins (stocks), F_i the total contents of bin i, and C the maximum contents of a bin. k is the parameter that defines how much stress we put on the nominator of the formula (the filling of the bins) as opposed to the denominator (the total number of bins). Setting k to 1 comes down to using the inverse of the number of bins. By increasing k, we give a higher fitness to solutions that contain a mix of well-filled and less well-filled bins, rather than equally filled bins. Falkenauer and Delchambre report that a value of 2 seems to be optimal. Values of more than 2 can lead to premature convergence, as the fitness of suboptimal solutions can come too close to the fitness of optimal solutions. In [11] Falkenauer proves algebraically that for k-values of more than 2, a solution of N+1 bins with N_F full bins could get a fitness higher than a solution with N equally filled bins.

4.5 Updating the pheromone trail

For the updating of the pheromone trail, we mainly followed the approach of Stützle and Hoos's MAX-MIN Ant System (MMAS) ([20]). We chose this version of the ACO algorithm because it is simple to understand and implement, and in the same time gives very good performance.

In MMAS, only the best ant is allowed to place pheromone after each iteration. We adapted equation 2 to reflect this. The equation is further changed because of the nature of the BPP and the CSP: as mentioned before, item sizes i and j are not unique, and they might go together several times is the bins of the best solution. We will increase $\tau(i,j)$ for every time i and j are combined. So finally, we get equation 7 below. In this equation, m indicates how many times i and j go together in the best solution s^{best}.

$$\tau(i,j) = \rho.\tau(i,j) + m.f(s^{best}) \tag{7}$$

Using only the best ant for updating makes the search much more aggressive. Bin combinations which often occur in good solutions will get a lot of reinforcement. Therefore, MMAS has some extra features to balance exploration versus exploitation. The first one of these is the choice between using the iteration-best ant (s^{ib}) and the global-best (s^{gb}). Using s^{gb} results in strong exploitation, so we will alternate it with the use of s^{ib}. We use a parameter γ to indicate the number of updates we wait before we use s^{gb} again.

Another way of enhancing exploration is obtained by defining an upper and lower limit (τ_{max} and τ_{min}) for the pheromone values (hence the name MAX-MIN). Stützle and Hoos define the value for the upper and lower limit algebraically. In our approach, we can't use an upper limit. This is because, depending on how many times item sizes appear together in the good solutions (so m in equation 7), pheromone values get reinforced more, and they evolve to different values. We would have to use different maximum values for different entries in the pheromone matrix.

We do use the lower limit τ_{min}, though. Stützle and Hoos calculate the value for τ_{min} based on *pbest*, the probability of constructing the best solution found when all the pheromone values have converged to either τ_{max} or τ_{min}. An ant constructs the best solution found if it adds at every point during solution construction the item with the highest pheromone value. Starting from this, Stützle and Hoos find the following formula for τ_{min} (see [20] for details):

$$\tau_{min} = \frac{\tau_{max}.(1 - \sqrt[n]{pbest})}{(avg - 1).\sqrt[n]{pbest}} \tag{8}$$

In this equation is n the total number of items, and avg the average number of items to choose from at every decision point when building a solution, defined as $\frac{n}{2}$. In our approach, we used this formula, but replaced τ_{max} in it by $\frac{1}{1-\rho}$. This is in fact an approximation of τ_{max} as calculated by Stützle and Hoos for values for m of 0 or 1, replacing the fitness of the best solution by 1[3]. The result is equation 9. The fact that several combinations of the same items are possible interferes quite severely with the calculations to get to equation 8, and the value of *pbest* should therefore only be seen as a crude approximation of the real probability to construct the best solution.

$$\tau_{min} = \frac{\frac{1}{1-\rho}.(1 - \sqrt[n]{pbest})}{(avg - 1).\sqrt[n]{pbest}} \tag{9}$$

A last feature we take over from MMAS is the pheromone trail initialisation. By starting from optimistic initial values, MMAS offers yet another way to enhance exploration. Stützle and Hoos put the initial pheromone values $\tau(0)$ to τ_{max}. We defined the value for $\tau(0)$ experimentally (see section 5.1).

5 Experimental Results

This section describes the results of our experiments. First the different parameter values are defined, and then the algorithm is compared to Liang et Al.'s Evolutionary Programming approach ([16]), Martello and Toth's Reduction Algorithm ([17]), and Falkenauer's HGGA ([11]).

[3]For most problems, the fitness value of the optimal solution lies somewhere between between 0.95 and 1.

5.1 Defining Parameter Values

In our tests to define parameter values, we used test problems that can be found on Klein and Scholl's webpage at the Technische Universität Darmstadt[4]. We used problems of different sizes and structures in order to get as general results as possible.

The first parameter to define was *nants*. To choose its value, we ran tests with a fixed number of solution constructions, but different number of ants. Apparently, a number of ants equal to the number of items in the problem gave the best results for all problems.

The next parameter, β, is the one that defines the relative importance of the heuristic information as opposed to the pheromone information. From our findings, this parameter appeared to be crucial. Using the wrong value for it resulted inevitably in bad results. However, we could not find a link between any features of the problem instance and the best value for beta. Fortunately, the good beta values for the different problems were all situated between 2 and 10, and in practice, the choice could be narrowed down to one of 2, 5 or 10. This means, though, that for every new problem these three values have to be tried out.

For the parameter k, which defines the fitness function, the results of Falkenauer ([11]) and Falkenauer and Delchambre ([12]) could be confirmed. A value of 2 was definitely better than 1. Higher values gave slightly worse results.

The parameters ρ, the pheromone evaporation, and γ, defining when updates have to be done with s^{gb} rather than s^{ib}, appeared to be interdependent. When examined separately, they both depended on the problem size. Once γ was set to $\lceil \frac{500}{n} \rceil$ (with n being the number of items in the problem), ρ had one optimal value for every problem: 0.95.

The optimal value for *pbest*, which defines τ_{min}, appeared to be 0.05, although a really broad range of values could be used, and the tests were not very conclusive. Also for $\tau(0)$, the initial pheromone value, a broad range of values gave good results, although setting $\tau(0)$ to τ_{min} (and giving up on optimistic initial values) gave clearly worse results. We chose to set it to $\frac{1}{1-\rho}$, which is an approximation of τ_{max} as defined by Stützle and Hoos.

[4]url: http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/binpp/

5.2 Comparing to other Approaches

We compared our approach for the CSP to Liang et Al.'s Evolutionary Programming approach (EP), and for the BPP to Martello and Toth's Reduction Algorithm and Falkenauer's HGGA. All tests were run on Sun sparc machines: Ultra 5's and Blade 100's with 128Mb using 270-502MHz processors. The algorithm was implemented in java.

5.2.1 Tests for the CSP

Liang et Al. include in their paper ([16][5]) their 20 test problems. We use their 10 single stock length problems (problem 1a to 10a) to compare our approach to theirs. They have a version of their program with and without contiguity. A CSP with contiguity is one where, apart from minimising the number of stocks, you also want as few outstanding orders as possible. Concretely, this means that once you have started cutting items of a certain length, you want to finish all the items of that length as soon as possible. Liang et Al's EP with contiguity gives the best results in number of stocks, so that will be the one we compare to.

Like Liang et Al., we did 50 independent test runs for each problem. The results are summarised in table 1. Liang et Al. use a population size of 75 and a fixed number of generations for each problem. In order to get a fair comparison, we let our ant algorithm maximally build the same total number of solutions as their EP: we multiplied the number of generations with the population size, and divided that by the number of ants (dependent on the problem size) to get the maximum number of cycles. Only for problem 10a we allowed less solutions (the same number as problem 9a), because the runs would otherwise take too long. As mentioned before, the parameter β is really crucial in our algorithm. Therefore, we had to do a few preliminary test runs for every problem to choose a good β value.

It is clear from these results that problems 1a to 5a were too easy: both algorithms find the best solution very quickly. For the other 5 problems, it is clear that our ACO algorithm finds better results: apart from problem 7a, the ACO algorithm finds both better average values and better best values. In fact, t-tests show that the EP results are less good with 100% probability for these problems. For problem 7a, the EP results are less good with 93.7%.

[5]This paper can be found at http://citeseer.nj.nec.com/385325.html

Prob	ACO				EP	
	avg	opt	cyc	time	avg	opt
1a	9	9	1	0	9	9
2a	23	23	1	0	23	23
3a	15	15	3	0	15	15
4a	19	19	9	1	19	19
5a	53	53	31	14	53	53
6a	79	79	198	2421	80.76	80
7a	68.82	68	385	1619	68.96	68
8a	144.92	144	366	6443	148.08	147
9a	150.98	150	345	20401	152.42	152
10a	218.44	218	442	14926	220.28	219

Table 1: These are the results for problem 1a up to problem 10a. 'ACO' gives the results obtained with our ACO approach, 'EP' gives Liang et Al.'s results. 'avg' indicates how many stocks were used on average, 'opt' indicates the number of stocks in the best solution, 'cyc' indicates after how many cycles this number of stocks was first found, and 'time' indicates the average running time in CPU seconds.

The table also shows an important disadvantage of our approach, though: it is quite slow. This is especially a problem for the big problems (6a to 10a). For 9a, every run takes on average almost 6 hours. This could probably be reduced a lot if the program was re-implemented in C, and run on faster machines (experience shows that it would be possible to get a speedup of 25). Also, the maximum number of cycles could be reduced (now it is 750000 solutions / 400 ants = 1875), as most results were obtained much earlier (on average after 345 cycles).

5.2.2 Tests for the BPP

In [11], Falkenauer compares his HGGA to Martello and Toth's Reduction Algorithm (MT). He uses 8 different sets of 20 test problems. The first four sets contain problems with a bin capacity of 150 and item sizes uniformly distributed between 20 and 100. He uses four different problem sizes: 120 items, 250, 500 and 1000. For each size, 20 different problems were created randomly. The next four sets of test problems have a different structure. They are the so-called triplets. This name is derived from the fact that in the optimal solution, every bin contains three items, of which two are smaller than the third of the bin capacity, and one is larger. These problems are very hard, because it is possible to fit three small items in a bin, or two large ones, but then you will inevitably miss the optimum. Again, 20 different problems were created for four dif-

Prob	HGGA		MT		ACO	
	bins	time	bins	time	bins	time
u120	0.10	381	0.10	370	0.20	370
u250	0.15	1337	0.60	1516	0.65	2476
u500	0.00	1015	2.20	1535	1.35	23378
t60	0.10	47	1.55	313	1	307
t120	0.00	79	4.10	921	1	1419
t249	0.00	728	7.45	2074	2	12265
t501	0.00	1663	14.85	5020	5.80	15416

Table 2: This table summarises the results of Falkenauer (HGGA), Martello and Toth (MT) and our ACO algorithm (ACO) per problem set. 'u120' to 'u500' are the uniform problem sets, and 't60' to 't501' the triplets. 'bins' indicates how far on average the solutions were above the theoretical optimum. 'time' gives the average running time in CPU seconds.

ferent problem sizes: 60, 120, 249 and 501 (with an optimal solution of 20, 40, 83 and 167 bins respectively). All of these test problems are available on-line at the OR-library: http://mscmga.ms.ic.ac.uk/info.html.

We ran our ACO algorithm on each instance of every problem set, except for the fourth (uniformly distributed with size 1000). This is because these problems are too big to be solved within a reasonable amount of time with our approach. To define the maximum number of solutions, we followed the same approach as before: generations times population size. The number of generations Falkenauer used was 2000 for the two smallest uniform problem sets, 5000 for the for the larger uniform problems, 1000 for the smaller triplets, and 2000 for the larger triplets. The population size is 100 every time. We did all the test runs for β values of 2, 5 and 10. For the smaller problems (uniform 120 and triplet 60), the value for β did not really matter, but for the larger ones, it made a big difference. In the table below, we only report the results for the best β value. The results are summarised per problem set in table 2.

From the results for the uniform problems ('u120' up to 'u500'), it is clear that our algorithm cannot beat Falkenauer's HGGA. For the small problems (size 120) it does almost equally well, but for the largest (size 500), it is always a bit less good. This is somehow to be expected though, as Falkenauer's approach combines a meta-heuristic with local search, whereas ours is only a meta-heuristic. The MT also does slightly better than ACO on the smallest problems, but for the larger ones, it does worse. The disadvantage of our algorithm, namely that it is slow for big problems, can again be observed here.

The hardness of the triplet problems ('t60' up to 't501') seems to make the observations we could make for the uniform problems much more clear. Again, HGGA is the best. MT's results are comparable to the ACO's for the smallest problem, but as the problem size increases, it becomes clear that MT does much worse than our ACO approach. An interesting fact is that our approach has a very constant performance: although it never finds the optimum, it is always close to it. MT sometimes finds the optimum for the smallest problems, but when it does not find the optimum, it often ends up in a very bad local optimum.

6 Conclusions

We have presented an ACO approach for the Bin Packing Problem and the Cutting Stock Problem. Artificial ants build solutions stochastically, using heuristic information and an artificial pheromone trail. The entries in the pheromone trail matrix encode the favourability of having two items in the same bin, and are reinforced by good solutions. The relative importance of the pheromone trail information as opposed to the heuristic information is defined by the parameter β, and is crucial for the performance of the algorithm. Unfortunately, there does not seem to be a link between the optimal value for this parameter and the features of the problem. It has to be defined empirically.

When compared to other approaches, it seems to be able to outperform Liang et Al.'s EP for the CSP and Martello and Toth's Reduction Algorithm for the BPP. It fails, however, to compete with Falkenauer's HGGA, which combines GA's with local search and is at the moment the best solution method for the BPP. A disadvantage of our method seems to be that it is very slow. A good point is that it has a steady performance, and when it does not find the optimum, it still finds a good solution.

We are currently trying to improve our program by combining it with a simple local search algorithm, based on Martello and Toth. It is a known fact (see [9]) that a local search algorithm can greatly improve the performance of an ACO approach. The first results we obtained suggest that the hybrid method is faster (thereby solving an important short-coming of our algorithm) and finds better solutions. In future work, it would also be interesting to extend the algorithm to cope with multiple stock lengths and contiguity (something Liang et Al.'s EP is capable of).

Acknowledgements

We would like to thank Ko-Hsin Liang and Xin Yao for sharing their results with us.

References

[1] George Bilchev. Evolutionary metaphors for the bin packing problem. In L. Fogel, P. Angeline, and T. Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 333–341, Cambridge, MA, USA, 1996. MIT Press.

[2] E. E. Bischoff and G. Wäscher. Cutting and packing. *European Journal of Operational Research*, 84:503–505, 1995.

[3] Eric Bonabeau, Marco Dorigo, and Guy Theraulez. *Swarm Intelligence: from natural to artificial intelligence*. Oxford University Press, Inc., New York, NY, USA, 1999.

[4] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[5] E. G. Coffman, M. R. Garey, and D. S. Johnson. *Approximation Algorithms for Bin Packing: A Survey*, pages 46–93. PWS Publishing, Boston, MA, USA, 1996.

[6] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

[7] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

[8] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics B*, 26(1):1–13, 1996.

[9] Marco Dorigo and Thomas Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. to appear in Handbook of Metaheuristics, F. Glover and G. Kochenberger.

[10] Harald Dyckhoff. A typlology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[11] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[12] Emanuel Falkenauer and Alain Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the IEEE 1992 International Conference on Robotics and Automation*, Nice, France, May 1992.

[13] L. M. Gambardella, E. D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167–176, 1999.

[14] Robert W. Haessler and Paul E. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150, 1991.

[15] Robert Hinterding and Luftar Khan. Genetic algorithms for cutting stock problems: with and without contiguity. In X. Yao, editor, *Progress in Evolutionary Computation*, pages 166–186, Berlin, Germany, 1995. Springer.

[16] Ko-Hsin Liang, Xin Yao, Charles Newton, and David Hoffman. A new evolutionary approach to cutting stock problems with and without contiguity, 2001. To appear in Computers and Operations Research.

[17] Silvano Martello and Paolo Toth. *Knapsack Problems, Algorithms and Computer Implementations*. John Wiley and Sons Ltd., England, 1990.

[18] Colin Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[19] T. Stützle. An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing*, pages 1560–1564, Aachen, Germany, 1998. Verlag Mainz.

[20] Thomas Stützle and Holger Hoos. Max-min ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[21] Marco Vink. Solving combinatorial problems using evolutionary algorithms, 1997. url: citeseer.nj.nec.com/vink97solving.html.