# Ant Colony Optimisation for Bin Packing and Cutting Stock Problems

*Frederick Ducatelle*

Master of Science

School of Artificial Intelligence
Division of Informatics
University of Edinburgh

2001

# Abstract

The bin packing and the cutting stock problems are two well-known NP-hard combinatorial optimisation problems. Only very small instances can be solved exactly, so for real-world problems we have to rely on heuristic solution methods. In recent years, researchers have started to apply evolutionary approaches to these problems, including genetic algorithms and evolutionary programming. In this dissertation, I try to solve the bin packing and the cutting stock problem using ant colony optimisation, a new class of meta-heuristics introduced by Dorigo in 1992. This meta-heuristic is inspired by the path-finding abilities of real ant colonies. It combines an artificial pheromone trail with simple heuristic information to stochastically build new solutions. I show that this approach gives good results, especially when combined with local search, and that it can outperform existing evolutionary approaches. A disadvantage of the method is that it is quite sensitive to the relative weighing of the heuristic information as opposed to the pheromone trail information.

# Acknowledgements

In the first place I would like to thank my supervisor, John Levine, for all the invaluable help and advice he gave me. I would also like to thank Peter Ross for providing useful references, and Ko-Hsin Liang and Xin Yao for sharing their results.

Many thanks also to my friends and family for giving me support throughout this MSc.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Frederick Ducatelle*)

# Table of Contents

# Chapter 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 [Dyckhoff, 1990] for an overview). In the BPP, the aim is to combine items into bins of a certain capacity so as to minimise the total number of bins, whereas in the CSP, the aim is to cut items from stocks of a certain length, minimising the total number of stocks. Obviously these two problem classes are very much related, and the approach proposed in this work will be able to tackle both of them.

Exact solution methods for the BPP and the CSP can only be used for very small problem instances. For real-world problems, heuristic solution methods have to be used. Traditional solution methods for the BPP include fast heuristics ([Dyckhoff, 1990]) and the reduction algorithm of Martello and Toth ([Martello and Toth, 1990]). CSP instances are traditionally solved with sequential heuristics or methods based on linear programming ([Haessler and Sweeney, 1991]). In the ongoing search for better solution methods for both problem classes, researchers have recently shown a lot of interest for evolutionary approaches, such as genetic algorithms ([Falkenauer, 1996, Hinterding and Khan, 1995, Reeves, 1996, Vink, 1997]) and evolutionary programming ([Liang et al., 2001]). The most successful of these new approaches is Falkenauer's hybrid grouping genetic algorithm ([Falkenauer, 1996]), which combines a grouping based genetic algorithm with a simple local search inspired by Martello and Toth's work.

In this dissertation, I propose an Ant Colony Optimisation (ACO) approach to the

BPP and the CSP. ACO is a new meta-heuristic for combinatorial optimisation and other problems. The first ACO algorithm was developed by Dorigo as his PhD thesis in 1992, and published under the name Ant System (AS) in [Dorigo et al., 1996]. It was an application for the Travelling Salesman Problem (TSP), loosely based on the path-finding abilities of real ants. It uses a colony of artificial ants which stochastically build new solutions using a combination of heuristic information and an artificial pheromone trail. This pheromone trail is reinforced according to the quality of the solutions built by the ants. AS was able to find optimal solutions for some smaller TSP instances. After its first publication, many researchers have proposed improvements to the original AS, and applied it successfully to a whole range of different problems (see [Bonabeau et al., 1999] or [Dorigo and Stützle, 2001] 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 [Bilchev, 1996]).

Apart from a pure ACO approach, I also develop a hybrid ACO algorithm. This approach combines the ACO meta-heuristic with a local search algorithm similar to the one used by Falkenauer. Each ant's solution is improved by moving some of the items around, and the improved solutions are used to update the pheromone trail. The reason for trying such an approach is the knowledge that ACO and local search are complementary ([Dorigo and Stützle, 2001]). ACO performs a rather coarse-grained search, providing good starting points for local search to refine the results.

This dissertation is organised as follows. Chapter 2 introduces the two combinatorial optimisation problems, and describes the most important existing solution methods for them. Chapter 3 gives a general introduction to ACO algorithms, describing AS and some of its extensions and applications. Chapter 4 contains a detailed explanation of how we applied ACO to the BPP and the CSP, and how the approach was augmented with local search. Chapter 5 gives an overview of the experimental results. In that section, the ACO approaches are compared to Martello and Toth's reduction algorithm and Falkenauer's hybrid grouping genetic algorithm for the BPP and to Liang et Al.'s evolutionary programming approach for the CSP. Chapter 6 concludes with a summary of the project and an overview of possible future work on this subject.

# Chapter 2

# Packing Bins and Cutting Stock

This chapter introduces the combinatorial optimisation problems that are tackled in this dissertation: the bin packing problem (BPP) and the cutting stock problem (CSP). The first section contains a description of both problems. The second section gives an overview of traditional solution methods for the BPP, and the third does the same for the CSP. The fourth section describes some of the evolutionary approaches that exist for both problems.

## 2.1 The Problem Descriptions

The BPP and the CSP are two well-known NP-hard combinatorial optimisation problems. In the traditional one-dimensional BPP, a set $S$ of items is given, each of a certain weight $w_i$. The items have to be packed into bins of a fixed maximum capacity $C$. The aim is to combine the items in such a way that as few bins as possible are needed. In the traditional one-dimensional CSP, a set $S$ of items, each of a certain length $l_i$, is requested. These items have to be cut from stocks of a fixed length $L$. Again the aim is to combine the items in such a way that as few stocks as possible are needed.

From the above descriptions, it is clear that these two problems are very similar. They both belong to the large group of cutting and packing problems. Dyckhoff describes a common logical structure for this group of problems ([Dyckhoff, 1990]). There is always a set of small items and a stock of large objects. The aim is to combine the small items into patterns and assign the patterns to large objects. Other problems that follow this structure are for example the vehicle loading problem, the knapsack problem, the multiprocessor scheduling problem and even the multi-period capital budgeting problem.

In his work, Dyckhoff proposes a typology for cutting and packing problems. He distinguishes along four criteria. The first is the dimensionality: one-, two-, three- or N-dimensional problems. The second criterion is the kind of assignment: whether you want to place all the small items into a number of large objects of your choice (like in the BPP and the CSP), or you have a fixed number of large objects, and have to make an optimal selection from the small items (like in the knapsack problem[1]). The third criterion is the assortment of large objects: is there only one object, or are there several of the same figure, or are there objects of different figures. The last criterion is the assortment of small items: are there few items, or many items of many different figures, or many items of relatively few different figures, or do all the items have congruent figures.

When classifying the BPP and the CSP according to this typology, Dyckhoff only makes a distinction between them based on the last criterion, the assortment of small items. In the BPP there are typically many items of many different sizes, whereas in the CSP, the items are usually only of a few different sizes (so there are many items of the same size). This means that the difference between the two problem types is a rather subjective and gradual one. Still, this difference is important enough to dictate totally different solution approaches, as will become clear in the next two sections.

Bischoff and Wäscher ([Bischoff and Wäscher, 1995]) give a number of reasons why cutting and packing problems are an interesting topic of research. First, there is the applicability of the research: cutting and packing problems are encountered in many industries, such as steel, glass and paper manufacturing. Additionally, as pointed out in [Dyckhoff, 1990], there are many other industrial problems that seem to be different, but have a very similar structure, such as capital budgeting, processor scheduling and VLSI design. A second reason is the diversity of real-world problems: even though

[1] In the knapsack problem you are given a set $S$ of small items, each of a certain weight $w_i$ and a certain benefit $b_i$. There is one large object of a fixed capacity $C$, and the aim is to make a selection from the small items to fill the large object so that the total sum of benefits is optimal.

cutting and packing problems have a common structure, there can be a lot of interesting differences between them. A last reason is the complexity of the problems. Most cutting and packing problems are NP-complete. This is definitely the case for the traditional one-dimensional BPP and CSP, which are studied in this dissertation. Exact optimal solutions can therefore only be found for very small problem sizes. Real world problems are solved using heuristics, and the search for better heuristic procedures stays a major research issue in this field.

In this dissertation, only the traditional one-dimensional BPP and CSP are considered. As mentioned above, there are many interesting variants possible. The problem can for example have multiple stock or bin sizes. Also, it is possible to have multiple objectives. An example of this is the CSP with contiguity (see [Liang et al., 2001]). In a CSP with contiguity, you want to minimise the number of stocks and in the same time the number of outstanding orders. In concrete, 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. Another interesting extension is the case with multiple stock lengths where the different types of stock are available at different locations, so that there is a freight cost associated with each stock length (see [Haessler and Sweeney, 1991]). Finally, there is also a lot of interest in two- and three-dimensional BPP's and CSP's.

## 2.2 Traditional Solution Methods for the BPP

BPP instances are usually solved with fast heuristic algorithms. The best of these is first fit decreasing (FFD). In this heuristic, the items are first placed in order of non-increasing weight. Then they are picked up one by one and placed into the first bin that is still empty enough to hold them. If no bin is left the item can fit in, a new bin is started. This algorithm is described in figure 2.1. Another often used fast heuristic is best fit decreasing (BFD). This heuristic is described in figure 2.2. The only difference with 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 $\frac{11}{9}OPT+4$, in which $OPT$ is the number of bins in the

1. Sort the items in order of non-increasing weight.
2. Remove the first item, and place it in the first bin that has enough space left to hold it. If no bin is empty enough, start a new bin.
3. Repeat step 2 until all items are placed in a bin.

Figure 2.1: The First Fit Decreasing algorithm.

1. Sort the items in order of non-increasing weight.
2. Remove the first item, and place it in the best-filled bin that still has enough space left to hold it. If no bin is empty enough, start a new bin.
3. Repeat step 2 until all items are placed in a bin.

Figure 2.2: The Best Fit Decreasing algorithm.

optimal solution to the problem ([Coffman et al., 1996]).

Apart from these fast algorithms, the BPP can also be solved with Martello and Toth's Reduction Algorithm (RA) ([Martello and Toth, 1990]). This is slower (certainly for bigger problems), but gives excellent results. The basis of RA is the notion of dominating bins: when you have two bins $B_1$ and $B_2$, and there is a subset $\{i_1, \dots, i_l\}$ of $B_1$ and a partition $\{P_1, \dots, P_l\}$ of $B_2$, so that for each item $i_j$, there is a smaller or equal corresponding partition $P_j$, then $B_1$ is said to dominate $B_2$ (see figure 2.3). This means that a solution which contains $B_1$ will not have more bins than a solution containing $B_2$. The RA 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.

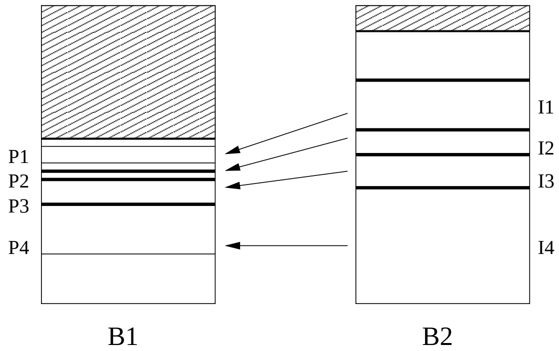


Figure 2.3: $B_2$ dominates $B_1$. Figure after [Falkenauer, 1996].

## 2.3 Traditional Solution Methods for the CSP

As described before, the difference between the BPP and the CSP only lies in the assortment of small items: in a BPP the items are usually of many different sizes, whereas in a CSP, the items are only of a few different sizes. This means that for a CSP, there is a structure in the demand: the same pattern of small items can be used several times to cut stock. So it makes sense to solve the problem in two steps: first build patterns, and then decide how many times to use each pattern. Traditional solution methods for the CSP follow this approach.

Two types of heuristic solution methods can be distinguished: linear programming (LP) based procedures and sequential heuristics. Most of the LP-based methods are inspired by the column generation method developed by Gilmore and Gomory in 1961 ([Gilmore and Gomory, 1961]). This method is based on the LP-relaxation of the problem:

$$\begin{array}{ll} \text{Minimise} & \sum_j X_j \\ \text{Subject to} & \sum_j A_{ij} X_j \geq R_i \quad \text{for all i} \\ & X_j \geq 0 \end{array} \tag{2.1}$$

Variable $X_j$ indicates the number of times pattern $j$ is used. $A_{ij}$ indicates how many times item $i$ appears in pattern $j$, and $R_i$ is the requested number of item $i$. So there are $i$ constraints indicating that for each item the demand has to be met. When solving an LP like this, one can also find the shadow price $U_i$ of each constraint $i$. The shadow price of a constraint indicates how much the goal function could be decreased if the right-hand side of that constraint would be relaxed by one unit. So, because constraint $i$ indicates the demand requirements for item $i$, its shadow price $U_i$ in fact indicates how much difficulties the algorithm has to reach the item's demand with the patterns considered so far. This information is then used in an integer programming model to make a new pattern (equation 2.2). The goal of this model is to fill the stock length with items while maximising the total benefit this will give to the LP model (indicated by the shadow prices $U_i$).

$$\begin{array}{ll} \text{Maximise} & \sum_i U_i A_i \\ \text{Subject to} & \sum_i L_i A_i \leq L \\ & A_i \geq 0 \qquad A_i \text{ is an integer} \end{array} \tag{2.2}$$

In this equation, $A_i$ indicates the number of times item $i$ is used in the pattern, $L_i$ is the length of item $i$ and $L$ is the stock length. The newly generated pattern is then again used to solve the LP-model of equation 2.1. More details about this can be found in [Haessler and Sweeney, 1991] and [Winston, 1993].

An alternative for these LP-based solution methods are the sequential heuristic procedures (SHP). They construct a solution by making one pattern at the time until all order requirements are satisfied. When making a pattern, other goals than waste minimisation can be taken into account. This is an advantage over LP approaches. There are also hybrid procedures possible, where an SHP is combined with an LP. For more details about this, see [Haessler and Sweeney, 1991].

## 2.4 Evolutionary Approaches

In recent years, people have tried various sorts of evolutionary approaches for the BPP and the CSP (e.g. see [Falkenauer and Delchambre, 1992, Hinterding and Khan, 1995, Reeves, 1996, Vink, 1997]). This section gives a short introduction to Falkenauer's hybrid grouping genetic algorithm (HGGA) for the BPP ([Falkenauer, 1996]) and Liang et Al.'s evolutionary programming approach (EP) for the CSP ([Liang et al., 2001]), because these are the algorithms the ACO approach of this project is compared to in chapter 5.

Falkenauer's HGGA is one of the most successful solution methods around for the BPP. It uses a grouping approach: the genetic algorithm (GA) works with whole bins rather than with individual items (for a complete example, see figure 2.4). When performing crossover, two crossover points are chosen. The bins of the second parent between the crossover points are inserted into the first parent, at its first crossover point. Obviously, there will be bins with overlapping items. These bins are deleted from the new solution, and their non-overlapping items become free. Before these items are re-injected into the solution, a simple local optimisation is performed (hence it is a hybrid GA). This local search is inspired by Martello and Toth's ideas about dominating bins: considering one by one the bins in the solution, the algorithm tries to replace up to three items in a bin by one or two of the free items in such a way that the total content of the bin increases without exceeding the maximum capacity. In this way, existing bins are made fuller, and free items are made smaller, so they are easier to fit into the bins. When this local optimisation phase is finished, the remaining free items are re-inserted into the solution using the FFD heuristic. The mutation operator works more or less in the same way: a few bins are eliminated at random and their items become free. These free items are again used for local optimisation, and the remaining items are re-inserted with the FFD heuristic. HGGA works extremely well and manages to outperform Martello and Toth's Reduction Algorithm. I am not sure how well it would work for the CSP, however. HGGA works with item numbers instead of item sizes. For the BPP, most items have a unique size, but in the CSP, many items can have the same size, so items with different numbers will in fact be the same.

Compared to all this, Liang et Al.'s EP for the CSP is a very simple algorithm. It is partly based on experiences by Hinterding and Khan ([Hinterding and Khan, 1995]). In that work, both a grouping GA and an order-based GA are used. Hinterding and Khan report that the grouping GA works better than the order-based GA, and that the performance of the order-based GA degrades when crossover is applied. Therefore, Liang et Al. use an order-based approach without crossover (see figure 2.5 for a complete example). Order-based means that solutions are represented within the EP as permutations of the items. Liang et Al. work with item sizes instead of item numbers, as their solution is aimed at the CSP where many different items can have the same size.

| | |
|---|---|
| HGGA works with item numbers instead of sizes: | |
| Item numbers: | $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ |
| The chromosomes are expressed in bins: | |
| The first parent: | $\{0, 1\}\{2\}\{3, 4, 5\}\{6, 7, 8\}\{9, 10\}$ |
| The second parent: | $\{0, 2\}\{4, 7\}\{5, 9\}\{1, 3, 8\}\{6, 10\}$ |
| Two crossover points are chosen: | |
| For the first parent: | $\{0, 1\}\{2\} \mid \{3, 4, 5\}\{6, 7, 8\} \mid \{9, 10\}$ |
| For the second parent: | $\{0, 2\} \mid \{4, 7\}\{5, 9\} \mid \{1, 3, 8\}\{6, 10\}$ |
| The bins between the second parent's points are entered at the first parent's first point: | |
| The child: | $\{0, 1\}\{2\}\{4, 7\}\{5, 9\}\{3, 4, 5\}\{6, 7, 8\}\{9, 10\}$ |
| Bins with double items are eliminated and their items which were not double become free: | |
| The remaining bins: | $\{0, 1\}\{2\}\{4, 7\}\{5, 9\}$ |
| The free items: | $3, 6, 8, 10$ |
| The algorithm tries to replace items in the bins with free items to make the bins fuller. | |
| A possible result could be: | |
| Remaining bins: | $\{0, 3\}\{2\}\{4, 10\}\{5, 9\}$ |
| Free items: | $1, 6, 7, 8$ |
| Finally, the remaining items are put into the bins using FFD (see figure 2.1): | |
| A possible result: | $\{0, 3\}\{1, 2\}\{4, 10\}\{5, 7, 9\}\{6, 8\}$ |
| Mutation opens bins at random: | |
| Remaining bins: | $\{0, 3\}\{1, 2\}\{6, 8\}$ |
| Free items: | $4, 5, 7, 9, 10$ |
| The remaining items are again placed back into the solution with FFD: | |
| A possible final result: | $\{0, 3\}\{1, 2, 4\}\{6, 7, 8\}\{5, 9, 10\}$ |

Figure 2.4: An example of the working of Falkenauer's HGGA.

| | |
|---|---|
| The EP works with item lengths instead of item numbers: | |
| The items: | $3, 3, 4, 4, 5, 6, 6, 6, 6$ |
| A chromosome is represented as an ordered list of items: | |
| A chromosome: | 5 4 6 3 3 4 6 6 |
| Decoding is done by cutting every time just before exceeding the stock length: | |
| Decoding for stock length 12: | 5 4 \| 6 3 3 \| 4 6 \| 6 |
| Mutation swaps twice three items around: | |
| Swapping two items: | 5 3 6 4 3 4 6 6 |
| Swapping three items: | 5 6 6 4 3 4 6 3 |
| Swapping twice three items: | 6 6 6 5 3 4 4 3 |
| Decoding again: | 6 6 \| 6 5 \| 3 4 4 \| 3 |

Figure 2.5: An example of the working of Liang et Al.'s EP algorithm.

The decoding of a chromosome happens by going through the item list, and making a cut every time a stock size is matched or the available stock size is exceeded. Mutation in this EP happens by swapping elements around: every parent produces one child by swapping twice three elements around. After the new children are formed, the new population is selected from the whole set of parents and children. Liang et Al. formulate a version of their algorithm for CSP's with and without contiguity. Their program is also able to solve multiple stock length problems. When compared to Hinterding and Khan's grouping GA (their best algorithm), the EP always gives comparable or better results.

# Chapter 3

# An Introduction to Ant Colony Optimisation

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 was called Ant System (AS). It was an application to solve the travelling salesman problem (TSP), developed in 1992 by Marco Dorigo as his PhD thesis ([Dorigo, 1992]). AS became very popular after its publication in 1996 (see [Dorigo et al., 1996]). Many researchers have since developed improvements to the original algorithm, and applied them to a range of different problems (see [Dorigo and Stützle, 2001]).

This chapter gives an introduction to ACO algorithms. The first section contains details about the path-finding behaviour of real ants that forms the biological inspiration for these algorithms. The second section shows how this behaviour was implemented artificially in the original AS to solve the travelling salesman problem. The third section gives an overview of improvements that have been made to the original algorithm, and the last section shows a variety of other problems that have been tackled with ACO algorithms.

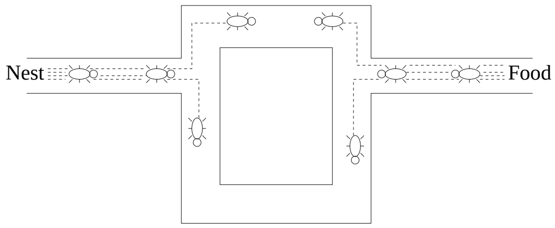


Figure 3.1: Real ants presented with a double bridge. Figure adapted from [Dorigo and Gambardella, 1997b].

## 3.1 The biological inspiration

ACO algorithms were originally inspired by the foraging behaviour of real ants. Many ant species leave a pheromone trail behind when walking between the nest and a food source, attracting in this way other ants to follow them. This process where an ant's path is influenced by the pheromone trail left behind by another ant is called recruitment.

Recruitment has a very interesting consequence, investigated by Deneubourg and colleagues in the late 1980's (a detailed description of their experiments can be found in [Bonabeau et al., 1999]). They placed a double bridge between a nest of Argentine ants and a food source, to investigate the trail-laying trail-following behaviour of the ants. The most interesting experiment was when they gave the bridges different lengths (see figure 3.1). The ants were left free to move and choose any of the two bridges, and in the beginning they chose randomly. In most experiments, however, it turned out that after a while all ants would end up using the shorter bridge. So even though individual ants have very limited vision and memory, the colony as a whole manages to find out which path is the shortest.

The key to this ability lies in the use of the pheromone trails. When the ants are first presented with the two bridges, they obviously don't know where to go, and they choose one at random. So we can assume that 50% of the ants choose the long bridge, and 50% the short bridge. It is clear, however, that ants using the short bridge will reach the food faster, and will be back faster. This means that after they have returned to the nest, the short bridge will contain more pheromone than the long one. When new ants have to make a decision, they will favour the short bridge, resulting in even more pheromone. And so, after a while, the whole colony will be going back and forth over the short bridge.

So the path-finding capabilities of the ant colony emerge from the behaviour of individual ants. The process is characterised by a positive feedback loop, in which ants are influenced to perform the same actions as others did before them. This kind of process that reinforces itself via positive feedback is called auto-catalytic (see [Dorigo et al., 1996]). The interesting aspect is that individually very limited agents are able to achieve something far beyond their own capabilities through a very simple form of cooperation as a group. The pheromone trail fulfils the task of collective memory, and guides the colony towards an optimal path.

## 3.2 Ant System

The first ACO algorithm was called Ant System (AS). It was developed by Marco Dorigo as his PhD thesis in 1992, and published in [Dorigo et al., 1996]. It was an application to solve the travelling salesman problem (TSP), inspired by the real ant behaviour described above. The original AS consisted of three different ant algorithms: *ant-density*, *ant-quantity* and *ant-cycle*. Most publications, however, identify AS with ant-cycle, the most successful of the three (see [Dorigo and Caro, 1999]). I will do the same, and refer the interested reader to [Dorigo et al., 1996] for details about ant-density and ant-quantity.

The TSP is defined as follows. There is a set of cities $C$, and a set of connections $L$, fully connecting the cities. With each connection between a city $i$ and a city $j$, a cost $\delta(i, j)$ is associated. The costs can be symmetric (the cost $\delta(i, j)$ is the same as $\delta(j,i)$) or asymmetric (in this case we speak of the asymmetric TSP). The goal of the TSP is to find a closed tour that visits all of the cities exactly once, while minimising the total cost of the connections used. It is clear that this problem bears some resemblance to the problem that the real ants have to solve: to find an optimal path. Apart from that, another reason why the TSP was chosen as the first ACO implementation, is that it is one of the most studied NP-hard problems in combinatorial optimisation.

The working of AS is based on the use of artificial pheromone: on top of its cost, each connection has an amount of pheromone $\tau(i,j)$ associated with it. This is a number that defines the desirability of a connection. Ants use this information probabilistically to build solutions, and update it afterwards.

AS works as follows. Each ant is placed in a randomly chosen initial city. Starting from there, it moves from city to city, building a solution to the TSP. When choosing the next city to move to, an ant considers only those cities it has not visited yet (so the artificial ants have a memory, unlike their natural counterparts). It chooses from those cities using the *random-proportional rule* given in equation (3.1). This rule gives the probability that ant $k$ in city $i$ chooses city $j$ next.

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)].[\eta(i,j)]^\beta}{\sum_{g \in J_k(i)} [\tau(i,g)].[\eta(i,g)]^\beta} & \text{if } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

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: $\frac{1}{\delta(i,j)}$. 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 $\beta$ that defines the relative importance of the heuristic information as opposed to the pheromone information. The role of the heuristic is to help the construction by providing problem specific information. $J_k(i)$ is the set of cities that have not been visited yet by ant $k$ in city $i$. The use of this set $J_k$ makes it actually very easy to implement constraints in AS: if certain choices would lead to an infeasible solution, you just exclude them from the set. Also, for very large problem instances, it is possible to narrow $J_k$ down by excluding choices that are certainly not optimal, thereby speeding up the algorithm significantly.

Once all ants have built a tour, the pheromone trail is updated. This is done according to the *global updating rule*:

$$\tau(i,j) = \rho.\tau(i,j) + \sum_{k=1}^{m} \Delta\tau_k(i,j) \tag{3.2}$$

$$\Delta\tau_k(i,j) = \begin{cases} \frac{1}{L_k} & \text{if } (i,j) \in \text{ tour of ant } k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Equation (3.2) consists of two parts. The left part makes the pheromone on all edges decay. The speed of this decay is defined by $\rho$, the evaporation parameter. The right part increases the pheromone on all the edges visited by ants. The amount of pheromone that 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 the quality of the solutions found by those ants.

It is easy to see the link between AS and reinforcement learning (for more information on this field, see [Sutton and Barto, 1998]): better solutions get a higher pheromone reinforcement, and new solutions are guided by this. Just like in reinforcement learning, it is important to balance exploration and exploitation. The AS algorithm described above leaves little opportunity for this. Most of the extensions of AS (which we will describe in the next section) focus exactly on doing this. In this way they try to improve the original AS, which performs well on small problems, but cannot compete with other algorithms on larger problems.

It is also easy to see the similarities with evolutionary algorithms (EA). Both approaches use a population of problem solutions, and use the information available in this population to stochastically build a new population of solutions. A main difference is that EA only use information available in the last population, whereas AS has a memory about earlier solutions in the pheromone trail. The one EA approach that is closest to AS is population based incremental learning (PBIL) ([Baluja and Caruana, 1995]). In PBIL, a vector with real numbers is kept, and it is used to stochastically generate a new population: every real number in the vector expresses the probability to generate a 1 rather than a 0 in that position in a new vector of bits for the population. So the role of this generating vector is similar to the role of the pheromone trail in AS. An important difference is the fact that in PBIL all the components of the generating vector are evaluated independently. This means that this approach only works well when the solution can be split into independent components.[1]

[1] This paragraph was mainly based on [Dorigo et al., 1999].

## 3.3 Extensions to Ant System

Many improvements to the original AS have been proposed. Most of these offer a stronger exploitation of previously found good solutions. They also give more opportunities to balance exploitation and exploration. This section contains a description of the most important extensions to AS. The focus is mainly on MAX-MIN Ant System, as that is the variant used in the rest of this dissertation.

The first improvement to AS was the elitist strategy proposed together with the original AS in [Dorigo et al., 1996]. It added to the normal pheromone trail updating of equation 3.2 an extra updating for the best solution found since the start of the algorithm. This greatly enhances exploitation. A similar approach can be found in $AS_{rank}$, a rank-based version of AS presented in [Bullnheimer et al., 1999]. There the pheromone is only updated for the best solution since the start of the algorithm (the global-best solution) and the $m$ best solutions of the last iteration (the iteration-best solutions). The updates of the iteration-best solutions are weighed according to their rank.

A third improved version of AS is Ant Colony System (ACS) (for details, see [Dorigo and Gambardella, 1997a] and [Dorigo and Gambardella, 1997b]). It is even more elitist than the previous two, as the updating is only done by the best ant. This causes strong exploitation, which is even enhanced by the fact that equation 3.1 is used deterministically with a probability $q_0$: with chance $q_0$, an ant $k$ in city $i$ deterministically chooses the city $j$ with the highest value for $p_k(i,j)$. With chance $1-q_0$, it uses $p_k(i,j)$ probabilistically to choose a city, like in AS. All this increased exploitation is balanced by local updating: every ant of the colony takes a bit of pheromone away from the branches it uses in its solution. In this way, following ants are less likely to build the same solution. ACS is based on Ant-Q (see [Dorigo and Gambardella, 1996]), which focused on the link with reinforcement learning. Ant-Q was more complicated than ACS without performing better.

The ACO algorithm which is mainly used in this project, is MAX-MIN Ant System (MMAS), proposed by Stützle and Hoos in [Stützle and Hoos, 2000]. For the building of a solution, MMAS follows equation 3.1 of the traditional AS. The difference with AS lies in the way the pheromone trail is updated. Like in ACS, only the best ant is allowed to update the pheromone trail. So equation 3.2 is replaced by equation 3.4. $\Delta\tau^{best}(i,j)$ in this equation is defined like $\Delta\tau_k(i,j)$ in equation 3.3.

$$\tau(i,j) = \rho.\tau(i,j) + \Delta\tau^{best}(i,j) \tag{3.4}$$

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 ($s^{gb}$) to do the updating. Using $s^{gb}$ results in strong exploitation, so usually it is alternated with the use of $s^{ib}$.

Another way of enhancing exploration is obtained by defining an upper and lower limit ($\tau_{max}$ and $\tau_{min}$) for the pheromone values (hence the name MAX-MIN). Stützle and Hoos define the value for the upper and lower limit algebraically. If a good solution is reinforced every time, the maximum value it could asymptotically obtain is given by equation 3.5 below (see [Stützle and Hoos, 2000] for mathematical details). $\tau_{max}$ is set to an estimate of this: the unknown length of the optimal solution $L_{s^{opt}}$ is replaced by the length of $s^{gb}$.

$$\frac{1}{1-\rho}.\frac{1}{L_{s^{opt}}} \tag{3.5}$$

The formula for $\tau_{min}$ is calculated based on $pbest$, the probability of constructing the best solution found when all the pheromone values have converged to either $\tau_{max}$ or $\tau_{min}$. 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 $\tau_{min}$ (see [Stützle and Hoos, 2000] for details):

$$\tau_{min} = \frac{\tau_{max}.(1-\sqrt[n]{pbest})}{(avg-1).\sqrt[n]{pbest}} \tag{3.6}$$

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 $\frac{n}{2}$.

A last way to enhance exploration in MMAS is by using optimistic initial pheromone values. The entries in the pheromone trail are initialised to $\tau_{max}$. This is a technique which is also often used in reinforcement learning (see [Sutton and Barto, 1998]). By setting all the pheromone values high in the beginning, the algorithm is forced to try out all different possibilities. After a while, the pheromone decay ensures that pheromone entries which are not often reinforced are reduced, and the exploratory effect wears out.

## 3.4 Applications of ACO algorithms

AS and its extensions were all first developed as applications for the TSP. This is mainly because of the similarities between this problem and the path-finding task that real ants solve. Later, many other problems have been tackled with ACO algorithms. This section describes some of the most interesting applications. A more complete overview of ACO applications can be found in [Bonabeau et al., 1999] or [Dorigo and Stützle, 2001].

The quadratic assignment problem (QAP) was after the TSP the second problem to be solved with an ACO approach. It was the first evidence of the robustness of AS. In an instance of the QAP, two $n*n$ matrices A and B are given. The aim is to find a permutation $\pi$ which minimises the following equation:

$$\min_{\pi \in \Pi(n)} f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}.b_{\pi_i \pi_j} \tag{3.7}$$

The first ACO solution method for the QAP, AS-QAP, was developed in 1994 ([Maniezzo et al., 1994]). Apart from the heuristic, the only difference with AS for the TSP is the pheromone trail definition. In AS-QAP, $\tau(i, j)$ indicates the favourability of setting $\pi_i$ to $j$. This approach performed reasonably well, but not extraordinarily when compared to other approaches. Later, in [Gambardella et al., 1999b], a different ACO approach for the QAP was proposed: HAS-QAP. This algorithm is very interesting because it completely moves away from the traditional ACO algorithms. Instead of building a new solution at every iteration, each ant maintains a solution, and updates it using the pheromone trail: at every iteration, the ants move the elements of their permutation around, guided by the pheromone trail. After this, every ant performs a simple local search to improve its solution. At the end of the iteration, the pheromone is updated. HAS-QAP performed extremely well on real-world problems, but less well on artificial unstructured problems.

After the QAP and the TSP, ACO solutions have been developed for many other combinatorial optimisation problems. Algorithms have been proposed for scheduling problems ([Bauer et al., 1999, Stützle, 1998]), the vehicle routing problem (VRP) ([Gambardella et al., 1999a]), the graph colouring problem ([Costa and Hertz, 1997]), the shortest common super-sequence problem ([Michel and Middendorf, 1999]), the multiple knapsack problem ([Leguizamon and Michalewicz, 1999]), and many other. Nevertheless, hardly any work has been done using ACO for the BPP and the CSP. In fact, the only publication related to this is a hybrid approach formulated by Bilchev ([Bilchev, 1996]). He uses ACO to combine a GA and a many-agent search model (MA) into one hybrid algorithm. Basically, a GA is run, and at the end of each of its generations, the $k$ best solutions are used to increase an artificial pheromone trail. Then this trail is used in an ACO algorithm to build $m$ new solutions. Finally, the MA starts from these solutions and tries to improve them, before updating the trail again. Bilchev's article is not very clear about implementation details. Also, no further research is done on this approach, as the aim was to develop, apply and compare different evolutionary metaphors, rather than to find a new solution method for the BPP. Bilchev's results do suggest that a model in which well-defined heuristics co-operate can outperform any of its composing algorithms.

ACO has also been used for problems other than combinatorial optimisation. An interesting application is AntNet, an algorithm for routing in packet-switched telecommunication networks (see [Caro and Dorigo, 1998]). The task is basically to find the minimum cost path between a pair of nodes in a network in which the costs are time-varying stochastic variables. To solve this task, AntNet associates with each directed arc $(i, j)$ in the network a set of pheromone values, one per possible destination node of the package. So $\tau(i, j, d)$ indicates the favourability to go from node $i$ to node $j$ for a packet with destination $d$. In order to define shortest paths between any pair of nodes in the network, ants are launched from each network node towards different destination nodes. The ants make their way through the network by choosing the next arc at every node probabilistically, based on the pheromone value for that arc and a heuristic value (which can be based on the length of the queue for this arc). Once an ant reaches its destination, it moves back along the same route it came by, and deposits pheromone according to the time it spent to reach the goal node. Then the pheromone is also decayed, like in AS. AntNet performs very well compared to other routing algorithms under a variety of network traffic conditions. An explanation for the good performance of AntNet in a time-varying environment like this can be found in the so-called "non-convergence" property of AS (see [Bonabeau et al., 1999]): even when an optimal solution is found, the population of solutions maintains a high diversity. This means that a wide sampling of the solution space is maintained at any time, which is favourable in dynamic environments.

# Chapter 4

# Applying ACO to the BPP and the CSP

This chapter describes how the ACO meta-heuristic was adapted to solve the BPP and the CSP. Section 1 explains how the pheromone trail was defined, section 2 describes which heuristic was used, section 3 talks about the fitness function that was used to guide the algorithm towards better solutions, section 4 shows how the pheromone trail is updated, and section 5 gives details about how the ants build solutions. After that, section 6 contains an explanation of how exploration can be increased by pheromone trail smoothing, and section 7 explains how local search was added to improve the performance of the algorithm. I will in this chapter and the next use the term “bin” when talking in general about the BPP and the CSP. Only when it clearly concerns a CSP I will use the term “stock”. A complete overview of the algorithm with and without local search can be found in appendix B, and the source code is available on-line at http://www.aiai.ed.ac.uk/ johnl/antbin.

## 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 ([Dorigo and Stützle, 2001]). It is crucial to choose a definition conform to the nature of the problem. The BPP and the CSP are grouping problems. What you essentially want to do, is divide 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

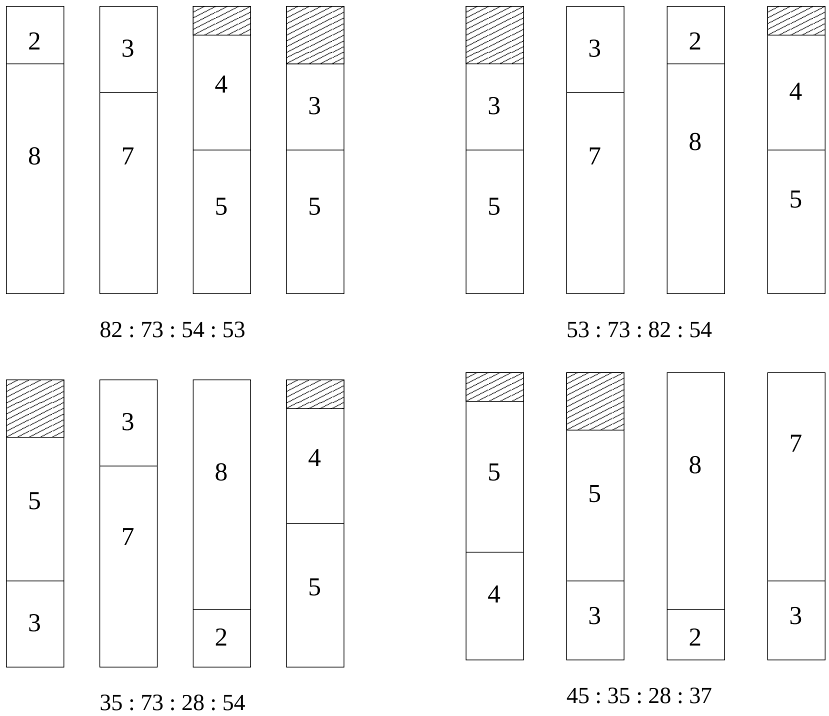


Figure 4.1: One solution can have many different order-based encodings.

problem: the aim is to put the different cities in a certain order. This is reflected in the meaning of the pheromone trail: it encodes the favourability of visiting a certain city $j$ after another city $i$.

It is possible to encode the BPP and the CSP as ordering problems. This was often done in GA solutions for these problems (see for example Liang et Al.'s approach described in section 2.4). A solution is then encoded as an ordered list of all items. A decoder is needed to translate this into a solution in terms of bins. As pointed out in [Falkenauer, 1996], however, there are a few problems with this. A first problem is redundancy. When you move the bins around in a solution to the BPP, or you move the items within a bin, you essentially keep the same solution. When encoded as an ordered list, however, all these presentations of the same solution become different encoded solutions (see figure 4.1). This means that there is a lot of redundancy in the solution space, and this redundancy grows exponentially with the problem size. A second problem with order-based encoding is the fact that the meaning of the place of

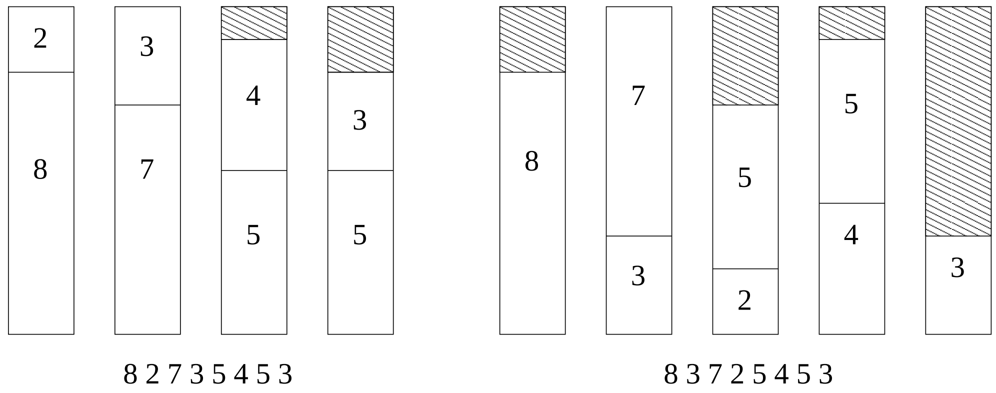


Figure 4.2: The meaning of an ordered-based encoding is context-dependent.

an item in the ordered list is context-dependent. The end of an ordered list can encode some interesting bins, but by changing the order of the items that precede it, this can be totally messed up. An example of this is given in figure 4.2: although the second half of the encoding is the same left and right, the result in the decoded solution is completely different.

Not everyone agrees that the theoretical arguments presented above really have any implications in practice ([Reeves, 1996, Vink, 1997]). In ([Hinterding and Khan, 1995]) on the other hand, an order-based GA is compared to a grouping GA, and it is found that the grouping GA always gives comparable or better results. It is also found that the performance of the order-based GA deteriorates with an increased crossover ratio. This is, according to Falkenauer, due to the context-dependency of the encoding. For the ACO algorithm in this project, I followed Falkenauer, and decided to use a grouping approach. It could be interesting to also try an order-based approach to compare this to.

To the best of my knowledge, there is only one ACO application for a grouping problem. It is Costa and Hertz's AntCol ([Costa and Hertz, 1997]), an ACO solution for the Graph Colouring Problem (GCP). In an instance of 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 (an example is given in figure 4.3). 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

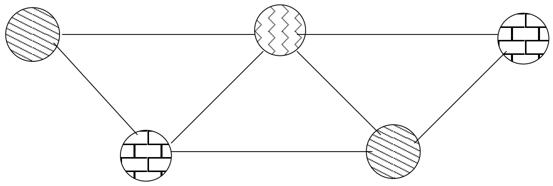

Figure 4.3: A simple example of a graph colouring problem.

favourability of having these nodes in the same colour. The pheromone matrix is of course symmetric ($\tau(i,j) = \tau(j,i)$).

In this project, the pheromone trail will be defined in the same way as in AntCol: $\tau(i,j)$ encodes the favourability of having an item of size $i$ and size $j$ in the same bin. 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 size $i$ and size $j$. As will become clear later, this has some important consequences for the ACO algorithm.

## 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. It guides the ants' probabilistic solution construction with problem specific knowledge. For the BPP and the CSP, a number of heuristics are available (see sections 2.2 and 2.3). As the heuristic has to be simple and easy to apply, it makes sense to choose one of the fast heuristics of the BPP. As described above, the best of these are FFD and BFD: they both give a guaranteed worst case performance of $\frac{11}{9}OPT + 4$ (with $OPT$ being the number of bins in the optimal solution to the problem). As FFD is simpler than BFD, it seems logical to use this heuristic for the ACO algorithm.

In FFD, the items are first sorted in order of non-decreasing weight, and then each item is placed in the first bin it still fits in (see figure 2.1 in section 2.2). Obviously, this heuristic will also work for the CSP. For use in the ACO algorithm, however, it will have to be reformulated. This is because the ACO approach works constructively: it

1. Open an empty bin.
2. Add the heaviest item that still fits in the bin.
3. Repeat step 2 until no item is left that is light enough to fit in the bin.
4. Go back to step 1 until all items are placed.

Figure 4.4: The reformulated First Fit Decreasing algorithm.

fills the bins one by one, closing one bin before starting the next, instead of the normal FFD approach, where the items are placed one by one while several half-filled bins can be open at the same time. The new FFD heuristic is described in figure 4.4: the bins are filled one by one with the heaviest items that can still fit in them. This results in the FFD solution, but is more useful for the ACO algorithm. The heuristic favourability of an item of size $j$ is now given by that size (the term size is used instead of weight to generalise the heuristic to the CSP):

$$\eta(j) = j \tag{4.1}$$

## 4.3 The Fitness Function

In order to guide the algorithm towards good solutions, it is important to be able to assess the quality of the solutions. So a fitness function is needed. A straightforward choice would be to take the inverse of the number of bins, so that better solutions get a higher fitness. As Falkenauer ([Falkenauer, 1996]) 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, I chose to use the function proposed by Falkenauer and Delchambre in [Falkenauer and Delchambre, 1992] to define the fitness of a solution $s$:

$$f(s) = \frac{\sum_{i=1}^{N}(F_i/C)^k}{N} \tag{4.2}$$

In this equation is $N$ the number of bins, $F_i$ the total contents of bin $i$, and $C$ the maximum contents of a bin. $k$ is the parameter that defines how much stress is 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 $k$, a higher fitness is given to solutions that contain a mix of well-filled and less well-filled bins, rather than equally filled bins. This forces the algorithm to look for item combinations that make full bins. Falkenauer and Delchambre report that a value of 2 for $k$ 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 [Falkenauer, 1996] it is proven 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.

Other researchers use this same fitness function ([Reeves, 1996, Vink, 1997]). A different function is used in the CSP applications of [Hinterding and Khan, 1995] and [Liang et al., 2001]. They use the formula of equation 4.3 below (in Hinterding and Khan's work, $N+1$ is replaced by $N$). This is a cost function, rather than a fitness function (it has to be minimised).

$$Cost = \frac{1}{N+1}\left(\sum_{i=1}^{N}\sqrt{\frac{F_i}{C_i}} + \sum_{i=1}^{N}\frac{V_i}{N}\right) \tag{4.3}$$

$N$ is the number of stocks used, $F_i$ is the total sum of the lengths of the items cut from stock $i$, $C_i$ is the length of stock $i$ (multiple stock lengths are possible), and $V_i$ is 1 if stock $i$ has waste (is not fully used) and 0 otherwise. The idea behind this formula is the same as behind the other fitness function: it simultaneously minimises the total waste and the total number of stocks with waste, so that solutions containing fully used stocks are favoured.

## 4.4 Updating the Pheromone Trail

For the updating of the pheromone trail, the main source of inspiration was Stützle and Hoos's MAX-MIN Ant System (MMAS) ([Stützle and Hoos, 2000]). This algorithm is described in detail in section 3.3. I chose this version of the ACO algorithm because it gives a very good performance (it is one of the best versions of the ACO heuristic), and in the same time is easy to understand and implement.

Like in MMAS, only the best ant is allowed to place pheromone after each iteration. The amount placed is given by the fitness of the solution built by that ant ($f(s^{best})$). Pheromone is placed for every two item sizes $i$ and $j$ that appear together in a bin of $s^{best}$. As item sizes are not unique, it is very well possible that $i$ and $j$ are combined more than once in $s^{best}$. In that case, $\tau(i,j)$ also gets more than one update. So equation 3.4 is adapted to get equation 4.4 below. In this formula is $m$ the number of times $i$ and $j$ go together in the bins of $s^{best}$.

$$\tau(i,j) = \rho.\tau(i,j) + m.f(s^{best}) \tag{4.4}$$

When talking about the best ant, one could be referring to the best ant of the past iteration (iteration best: $s^{ib}$), or the best ant so far in the algorithm (global best: $s^{gb}$). As mentioned in section 3.3, using only the best ant for pheromone updating results in a rather aggressive search. Therefore, it is a good idea to not exclusively use $s^{gb}$, but alternate it with $s^{ib}$. The parameter $\gamma$ indicates every how many times $s^{gb}$ is used. The value of this parameter is defined empirically (see chapter 5).

Another important feature of MMAS is the use of an upper and lower limit for the pheromone trail. The upper limit ($\tau_{max}$) is defined as an approximation of the asymptotic maximum a pheromone value can evolve to. In this ACO application for the BPP and the CSP, it is impossible to use an upper limit. This is because of the fact that the item sizes are not unique: if $i$ and $j$ go together twice in good solutions, and $i$ and $h$ once, then according to equation 4.4, $\tau(i,j)$ will get two updates every time, and $\tau(i,h)$ only one. This will finally result in an asymptotic maximum value for $\tau(i,j)$ which is the double of the maximum value for $\tau(i,h)$. It is of course impossible to know in advance how many times the items will go together, and therefore to define a good value for $\tau_{max}$.

The algorithm does use a lower limit for the pheromone trail ($\tau_{min}$). Stützle and Hoos define the value for $\tau_{min}$ based on $pbest$, the probability of constructing the best solution encoded in the pheromone trail when all pheromone values have converged to either $\tau_{max}$ or $\tau_{min}$. An ant constructs the best solution if it, at any point during the construction, chooses the item with the highest pheromone value. When a certain *pbest* is given, it is possible to calculate a value for $\tau_{min}$. Stützle and Hoos find formula 3.6. In that formula, I replaced the unknown $\tau_{max}$ by $\frac{1}{1-\rho}$. This is in fact an approximation of their formula for $\tau_{max}$, given in equation 3.5: as the fitness of the optimal solution is somewhere between 0.95 and 1 for most problems, I chose to approximate it by 1. The resulting equation for $\tau_{min}$ is:

$$\tau_{min} = \frac{\frac{1}{1-\rho}.(1-\sqrt[n]{pbest})}{(avg-1).\sqrt[n]{pbest}} \tag{4.5}$$

The fact that item size combinations can appear more than once in the best solution would interfere quite severely with the calculations to get to equation 3.6. Also, $\frac{1}{1-\rho}$, used in 4.5, is only a good approximation for $\tau_{max}$ if item sizes only appear 1 or 0 times together in a solution. Therefore, when equation 4.5 is used as $\tau_{min}$ in this ACO approach, *pbest* can only be seen as a very crude approximation of the real probability to construct the best solution. In fact, you can expect *pbest* to be further from the real probability of constructing the best solution the more items there are of the same size.

Finally, a last element we take over from MMAS is the high pheromone trail initialisation. This is meant to enforce exploration in the first few iterations of the algorithm: as all pheromone values are high in the beginning, all the different possibilities are tried out. After a while, through the pheromone decay, the pheromone entries which are not often used in the best solutions get less reinforcement and go down, and the exploratory effect of the high initial values wears out. Stützle and Hoos set the initial pheromone value $\tau(0)$ to $\tau_{max}$. For this project, $\tau(0)$ was defined empirically to $\frac{1}{1-\rho}$, the approximation of $\tau_{max}$ described above (see chapter 5).

## 4.5 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 be placed and an empty bin. It adds the items one by one to this bin, until none of the items left is small enough to fit in it. Then the bin is closed, and a new one is started. The probability that an ant $k$ will choose an item of size $j$ as the next item for its current bin $b$ in the partial solution $s$ is given by equation 4.6 below. This equation is very similar to the one used in the original AS (see section 3.2).

$$p_k(s,b,j) = \begin{cases} \frac{[\tau_b(j)].[\eta(j)]^\beta}{\sum_{g \in J_k(s,b)} [\tau_b(g)].[\eta(g)]^\beta} & \text{if } j \in J_k(s,b) \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

In this equation, $J_k(s,b)$ is the set of items that qualify for inclusion in the current bin $b$. They are the items that are still left after partial solution $s$ is formed, and are small enough to fit in bin $b$. $\eta(j)$ is the item size $j$, as defined in equation 4.1 above. The pheromone value $\tau_b(j)$ for an item size $j$ in a bin $b$ is given in equation 4.7 below. It is the sum of all the pheromone values between item size $j$ and the item sizes $i$ that are already in bin $b$, divided by the number of items in $b$ for normalisation. If $b$ is empty, $\tau_b(j)$ is set to 1. Like in other ACO approaches is $\beta$ the parameter that defines the relative importance of the heuristic information as opposed to the pheromone value. This whole approach is similar to the one followed by Costa and Hertz in their ACO application for the GCP (see section 4.1).

$$\tau_b(j) = \begin{cases} \frac{\sum_{i \in b} \tau(i,j)}{|b|} & \text{if } b \neq \{\} \\ 1 & \text{otherwise} \end{cases} \tag{4.7}$$

As was described in section 4.4, the pheromone value $\tau(i,j)$ between the item sizes $i$ and $j$ is increased for every time $i$ and $j$ occur together in a bin. This means that if, for example, sizes $i$ and $j$ occur $l$ times together in a solution, the pheromone value $\tau(i,j)$ between them will receive $l$ updates per iteration, whereas if sizes $i$ and $h$ occur only once together, $\tau(i,h)$ will receive only one update. This will after a while result in a value for $\tau(i,j)$ that is $l$ times higher than $\tau(i,h)$. I originally feared that this would give a problem, because the combination $(i,j)$ would always be favoured over $(i,h)$, not only $l$ times, but also the $l+1^{\text{th}}$ time. I tried to solve this by making changes to the pheromone trail while it was being used by the ants: every time an ant used a pheromone value, this value was lowered, so that next time, in another bin of the same solution, the ant would favour this combination of items less. In later tests, however, this turned out to be unnecessary: the algorithm worked better without the changes. This is because of the fact that equation 4.6 expresses a probability, rather than a deterministic indication of which item size to choose next. This means that every time an new item has to be chosen for a bin containing an item of size $i$, item size $j$ has $l$ times more chance than $h$ to be picked. So, on average, the combination $(i,j)$ will be constructed $l$ times more than $(i,h)$. And thus, on average, the resulting solution will be correct.

## 4.6 Pheromone Trail Smoothing

Pheromone trail smoothing was proposed in [Stützle and Hoos, 2000] as an additional mechanism to increase the performance of MMAS. The basic idea is to increase the pheromone values proportionally to their difference to the maximum pheromone value $\tau_{max}$. This is done when MMAS has converged or is very close to convergence (meaning that all pheromone values are really close to either $\tau_{max}$ or $\tau_{min}$). By doing this, the probability of choosing an item with low pheromone value is increased. This means there is enhanced exploration. The formula proposed by Stützle and Hoos is the following:

$$\tau(i,j) = \tau(i,j) + \delta.(\tau_{max} - \tau(i,j)) \text{ with } 0 < \delta < 1 \tag{4.8}$$

With a $\delta$ value of 1, the smoothing comes down to complete pheromone reinitialisation. With a $\delta$ of 0, smoothing is switched of. Usually, a $\delta$ between 0 and 1 is used, so that the information contained in the pheromone trail is weakened but not lost. As explained is section 4.4, $\tau_{max}$ is unknown in the ACO application for this project. We therefore replace it in equation 4.8 by $\tau(0)$. A disadvantage of pheromone trail smoothing is that it usually needs longer runs to be effective.

## 4.7 Adding local search

It is known that the performance of ACO algorithms can sometimes be greatly improved when coupled to local search algorithms ([Dorigo and Stützle, 2001]). This is for example the case in applications for the TSP, the QAP and the VRP. What normally happens is that a population of solutions is created using ACO, and then these solutions are improved via local search. The improved solutions are then used to update the pheromone trail. So it is in fact a form of Lamarckian search.

An explanation of the good performance of a combination of ACO with local search can be found in the fact that these two search methods are complementary. An ACO algorithm usually performs a rather coarse-grained search. Therefore, it is a good idea to try and improve its solutions locally. A local search algorithm, on the other hand, searches in the surroundings of its initial solution. Finding good initial solutions is however not an easy task. This is where ACO comes in: by generating new promising solutions based on previously found optima, the local search can be given very good starting points.

There are not so many local search algorithms around for the BPP or the CSP. One algorithm that seems to work fairly well was proposed in [Alvim et al., 1999]. In that algorithm, an initial solution is constructed using the BFD heuristic. Then each bin of the current solution is destroyed successively, and its contents are spread over the other bins. If this leads to a feasible solution (with no overflowing bins), we have obtained a solution with one bin less. If the spreading of the items leads to an infeasible solution, a local search is applied: pairs of bins are investigated and its items are redistributed among themselves. If this leads to a feasible solution, a new solution improvement phase is finished.

As Alvim et Al. report, this local search algorithm gives pretty good results. But for combination with an ACO algorithm, I needed something fast and simple. Therefore, the local optimisation algorithm used in Falkenauer's HGGA (see section 2.4) seemed to be a better choice (although it would be very interesting to see how an ACO combined with Alvim et Al.'s approach would perform). In this algorithm, a number of items of the initial solution are made free. In the mutation phase of the HGGA, this is done by opening a few randomly selected bins. Then the algorithm tries to replace up to three items in each of the existing bins of the solution by one or two of the free items, in such a way that the total content of the bin is increased without exceeding the maximum capacity. After all bins have been examined, the remaining free items are added to the solution using the FFD heuristic. This search is inspired by Martello and Toth's dominance criterion (see section 2.2), which essentially states that well-filled bins with large items are always preferable over less-filled bins with smaller items. In HGGA, the algorithm searches locally for dominant bins, by replacing items in the bins by larger free items. In the same time, the free items are replaced by smaller items from the bins, which makes it easier to place them back into the solution afterwards.

In the hybrid version of my ACO algorithm, every solution created by an ant is taken through a local optimisation phase. In this phase, the least filled bins are destroyed, and their items become free (the number of bins to be destroyed is defined empirically, see chapter 5). Then, for every remaining bin, it is investigated whether some of its current items can be replaced by free items so that the bin becomes fuller. The algorithm successively tries to replace two current items by two free items, two current items by one free item, and one current item by one free item. In the end, the remaining free items are re-inserted into the solution using the FFD heuristic. A complete example of the local search phase is given in figure 4.5. The pheromone is updated using the locally improved solutions.

The solution before local search (the bin capacity is 10):

| | | |
|---|---|---|
| The bins: | 3 3 3 \| 6 2 1 \| 5 2 \| 4 3 \| 7 2 \| 5 4 | |

Open the two smallest bins:

| | | |
|---|---|---|
| Remaining bins: | 3 3 3\| 6 2 1 \| 7 2\| 5 4 | |
| Free items: | $5, 4, 3, 2$ | |

Try to replace 2 current items by 2 free items, 2 current by 1 free or 1 current by 1 free:

| | | |
|---|---|---|
| First bin: | 3 3 3 $\rightarrow$ 3 5 2 | new free: $4, 3, 3, 3$ |
| Second bin: | 6 2 1 $\rightarrow$ 6 4 | new free: $3, 3, 3, 2, 1$ |
| Third bin: | 7 2 $\rightarrow$ 7 3 | new free: $3, 3, 2, 2, 1$ |
| Fourth bin: | 5 4 stays the same | |

Reinsert the free items using FFD:

| | | |
|---|---|---|
| Fourth bin: | 5 4 $\rightarrow$ 5 4 1 | |
| Rest in new bin: | 3 3 2 2 | |
| Final solution: | 3 5 2 \| 6 4 \| 7 3 \| 5 4 1 \| 3 3 2 2 | |

Figure 4.5: An example of the use of the local search algorithm.

# Chapter 5

# Experimental results

This chapter summarises the results obtained in experiments with the ACO approach for the BPP and the CSP. In the first section, the various parameters for the pure ACO algorithm and for the ACO algorithm with local search are examined, and values are defined for them. In the second section, the algorithms are compared to other approaches: to Liang et Al.'s EP algorithm ([Liang et al., 2001]) for the CSP and to Martello and Toth's RA ([Martello and Toth, 1990]) and Falkenauer's HGGA ([Falkenauer, 1996]) for the BPP. All three of these approaches are described in chapter 2.

## 5.1 Defining parameter values

This section describes how parameter values were defined for the pure ACO algorithm and the ACO algorithm enhanced with local search. In the tests to define parameter values, I used test problems that are available on Klein and Scholl's webpage at the Technische Universität Darmstadt[1]. Problems of different sizes and structures were used to get as general results as possible.

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

### 5.1.1 The pure ACO algorithm

Different parameter values had to be defined for the pure ACO algorithm: the number of ants ($nants$), the relative weighing of heuristic and pheromone information ($\beta$), the relative importance of the filling of the bins in the fitness function ($k$), the pheromone evaporation ($\rho$), the relative number of updates to be done with $s^{ib}$ as opposed to $s^{gb}$ ($\gamma$), and the probability of constructing the best solution ($pbest$), which defines $\tau_{min}$. When the algorithm is extended with pheromone trail smoothing (PTS), also the degree of smoothing $\delta$ has to be defined.

To define the value for $nants$, the algorithm was run on the different test problems with a fixed number of solution constructions. So when $nants$ was increased, the number of iterations was decreased to maintain this fixed number of solutions. In this way, it was possible to compare a high number of iterations with only a few ants to a lower number of iterations with more ants. The tests indicated that, like for other ACO algorithms and applications (see [Dorigo and Stützle, 2001]), the algorithm was quite robust to this parameter. Still, it could clearly be observed that the range of values of $nants$ that gave optimal results varied with the problem size. Setting $nants$ to the number of items $nitems$ gave optimal or near-optimal results for all test problems. It is interesting to see that the optimal number of ants was well above 1, so it is useful to use a colony of ants rather than one individual ant.

The next parameter, $\beta$, defines the relative importance of the heuristic information as opposed to the pheromone information when ants build a solution. This parameter appeared to be crucial. Using a wrong value for it resulted inevitably in poor results. Through tests with many different problem instances, I tried to find a link between the optimal $\beta$ value and problem features, such as the number of items, the number of different item sizes, the quality of the FFD solution (as this is the heuristic that is used), etc.. I failed to find any useful relation, however, and could only conclude that $\beta$ has to be defined empirically for any new problem instance. Fortunately, the good $\beta$ values for the different problems were all situated between 2 and 10, and in practice, the choice can be narrowed down to one of 2, 5 or 10. Values of 0 always gave bad results, so I could confirm the general statement of [Dorigo and Stützle, 2001] that heuristic information is important to direct solution construction. I also did tests without the pheromone trail, so that only heuristic information was used (but in a stochastic way). This gave results which were better than the deterministic use of FFD, but clearly worse than the ACO algorithm, confirming that the ACO algorithm does something useful.

For the parameter $k$, which defines the fitness function, it was clear that a value of 2 was better than 1. So the fitness function of [Falkenauer and Delchambre, 1992] gave better results than just using the inverse of the number of bins (see section 4.3). Values higher than 2 did not give significantly worse results. As the results were not better either, I chose to keep 2 as the value for $k$, keeping in mind the theoretical arguments of [Falkenauer, 1996] against higher values.

The two next parameters, the pheromone evaporation rate $\rho$ and the relative number of updates to be done with $s^{gb}$ $\gamma$[2], appeared to be interdependent. For both of these parameters, a higher value means more exploration of the solution space. When examined separately, they both needed a lower value (so less exploration) for larger problem instances. This seems to make sense: larger problems have more items, so more places to make stochastic decisions while building a solution. So for large problems, more different solutions will be sampled anyway, and the probability to build the best solution encoded in the trail ($pbest$) will be lower. So less extra exploration is needed. I decided to set $\gamma$ to $\lceil \frac{500}{nitems} \rceil$. Once this was done, there was enough size-dependent exploration, and $\rho$ could be set to one fixed value for every problem: 0.95.

The optimal value for $pbest$, which defines $\tau_{min}$, 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. Setting $\tau(0)$ to $\tau_{min}$ (and giving up on optimistic initial values) gave clearly worse results though. I chose to set it to $\frac{1}{1-\rho}$, the approximation of $\tau_{max}$ defined in section 4.4.

Finally, I extended the algorithm with PTS, and did tests to define the best value for $\delta$, the parameter which defines how much of the original pheromone value you keep (with a $\delta$ of 1, you loose all previous pheromone information, with a $\delta$ of 0, smoothing is switched off) (see section 4.6). The optimal value for $\delta$ turned out to be 0.4. With this value, PTS could improve the results for some of the test problems significantly.

[2] A $\gamma$ value of 1 means that all updates have to be done with $s^{gb}$, a value of $n$ means that every $n^{\text{th}}$ update has to be done with $s^{gb}$.

For most problems, however, the improvements were only marginal.

### 5.1.2 The ACO algorithm with local search

When the ACO is combined with local search, new parameter settings are needed. In this section, the parameters $nants$, $\beta$, $\rho$, $\gamma$, *pbest* and $\tau(0)$ are redefined. The parameter $k$ is kept on 2. One new parameter is introduced: *bins* indicates the number of bins that are opened to release the free items for the local search.

To define $nants$, the same kind of test as before was done: vary the number of ants, while the total number of solution constructions stays fixed. Like for the pure ACO algorithm, a rather wide range of values gave good solutions. The best values for $nants$ were lower, however, and less dependent on the problem size. It was possible to set the value of $nants$ to 10 for all problems. The fact that less ants are needed per iteration can be explained as follows. If no local search is used, interesting spots are only found when ants specifically build those solutions. With local search, however, every solution is taken to a near-by optimum in the solution space. Therefore, less ants are needed to get an equally good sampling of interesting solutions.

When investigating the $\beta$ parameter in the algorithm with local search, it turned out that using an optimal $\beta$ value became less important, and that most problems could in fact do with a value of 2. This was to be expected: as is explained in [Dorigo and Stützle, 2001], local search uses the heuristic information in a more direct way to improve solutions, and the importance of the heuristic information for the building of solutions diminishes. However, there were still some problem instances that needed a $\beta$ value of 5 or 10, so it stays necessary to experimentally choose a good value.

The fact explained above that local search focuses the investigation directly on the interesting spots of the solution space also means that less exploration is necessary. This was confirmed when the optimal values for $\gamma$ and $\rho$ were defined. For $\gamma$, the optimum appeared to be 1 for any problem size, meaning that all the updates are done with the globally best solution $s^{gb}$. So there is less exploration. For $\rho$, the test results were very unclear. For most problems, any value between 0 and 0.9 gave good results. A very low $\rho$ value means that the pheromone only lasts for one generation, so new solutions are only guided by the previous optimum. This results in a very aggressive search, and for some rather difficult problems, the optimum was found incredibly fast (in as few as 2 or 3 iterations, where for the pure ACO algorithm 300 was more the norm for these problems). It did, however, also cause the algorithm to get stuck in local optima from time to time. In the end, I settled for a $\rho$ of 0.75. This gave rise to longer runs (around 30 iteration for the problems mentioned above), but was less unstable in terms of convergence into local optima.

Also, for *pbest* and $\tau(0)$, less exploration was the key word. For different values of *pbest*, the results in number of bins stayed the same, but less cycles were needed for higher values. The best results were in fact obtained with *pbest* set to 1. This means that $\tau_{min}$ is set to 0: the lower limit on pheromone values is abandoned. Also for $\tau(0)$, the results hardly differed in number of bins. Therefore I decided to give up on the exploratory starts as well (to set $\tau(0)$ to 0). This means that the algorithm is not first forced to try out different possibilities, so it can get right down to business, and needs less iterations.

Finally, also the new parameter *bins*, the number of bins to be opened, had to be defined. This was quite difficult, as it depended very much on the problem instance at hand. Fortunately, for most problems the algorithm gave optimal results for quite a wide range of values for *bins*. Originally, there was no overlap between the ranges for the different test problems I used. Then I tried to turn around the order in which the local search tries to replace existing items with free items (originally, the algorithm first replaced 1 existing item for 1 free item, then 2 for 1 and finally 2 for 2). This gave slightly better solutions, and made the ranges of optimal *bins* values wider, so that a value of 4 became acceptable for all problems.

## 5.2 Comparing to other approaches

In this section, the pure ACO algorithm and the ACO algorithm augmented with local search are compared to existing evolutionary approaches for the CSP and the BPP. PTS was not used, as this technique only improves the results slightly, while using longer runs. For the CSP, the algorithms are compared to Liang et Al.'s EP approach. For the

BPP, they are compared to Martello and Toth's RA and Falkenauer's HGGA. All the tests were run on the department's Sun Sparc machines: Ultra 5's and Blade 100's with 128 Mb memory using 270-502 MHz processors. The algorithm was implemented in java.

### 5.2.1 Tests for the CSP

Liang et Al. include in their paper ([Liang et al., 2001]) their 20 test problems. I use their 10 single stock length problems (problem 1a to 10a included in appendix A) to compare my approach to theirs. They have a version of their program with and without contiguity. As is explained in section 2.1, a CSP with contiguity is one where, apart from minimising the number of stocks, you also want as few outstanding orders as possible. Liang et Al.'s EP with contiguity gives the best results in number of stocks. This could be due to the fact that the contiguity goal prefers a certain order of the items in the solution, regardless of their grouping into bins, so that an order based approach is favoured. It also reduces the redundancy in the solution space, which was said to be a major problem for order based approaches (see section 4.1). Even though my algorithms don't take contiguity into account, I will compare them to Liang et Al.'s EP with contiguity[3]. First, the EP is compared to the pure ACO algorithm. After that, it is compared to the ACO algorithm with local search.

Like Liang et Al., I did 50 independent test runs for each problem. The results for the pure ACO algorithm and the EP are summarised in table 5.1. 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, I let the ant algorithm maximally build the same total number of solutions as the EP: the number of generations was multiplied with the population size, and divided by the number of ants (dependent on the problem size) to get the maximum number of iterations. Only for problem 10a less solutions were allowed (the same number as for problem 9a), because the runs would otherwise take too long. As mentioned before, the parameter $\beta$ is really crucial in the ACO algorithm. Therefore,

[3]This algorithm differs in only two aspects from the one described in section 2.4: in 25% of the mutations, another operator is used (one that is better for contiguity), and a slightly different cost function is used (see [Liang et al., 2001] for details).

I had to do a few preliminary test runs for every problem to choose a good $\beta$ value. In the table, only the results for the best $\beta$ value are reported.

It is clear from these results that problems 1a to 5a were too easy: both algorithms always find the best solution. For the other 5 problems, the ACO algorithm finds better results: apart from problem 7a, it 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%. As mentioned above, however, the ACO algorithm only gave these good results when the best $\beta$ value was used. So, compared to the EP, it has the disadvantage that it needs a preliminary optimisation phase.

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

The results for the ACO algorithm with local search are summarised in table 5.2. As solution construction with this version of the algorithm obviously takes much more time, I had to reduce the maximum number of evaluations (see column 'sol' in the table). Since the number of ants was much lower than for the pure ACO algorithm, however (10 instead of the number of items in the problem), this did not result in less cycles. The parameter $\beta$ again had to be defined in a preliminary optimisation phase.

For problem 1a up to 6a, the results are, in terms of number of stocks, the same as for the pure ACO algorithm. For problems 1a up to 4a, these results are the theoretical optimum. For 5a and 6a, the theoretical optimum (found by summing the lengths of all items together and dividing this by the stock length) would be lower, but it is possible that that optimum cannot be reached in practice (as neither of the ACO algorithms, nor the EP can find it). For the bigger problems, 7a up to 10a, the ACO algorithm with local search performs clearly better than the pure ACO algorithm: it has a better average solution for all of these problems, and even finds a better best solution for

| Prob | ACO | | | | | EP | | |
|---|---|---|---|---|---|---|---|---|
| | sol | avg | opt | it | time | sol | avg | opt |
| 1a | 37500 | 9 | 9 | 1 | 0 | 37500 | 9 | 9 |
| 2a | 75000 | 23 | 23 | 1 | 0 | 75000 | 23 | 23 |
| 3a | 150000 | 15 | 15 | 3 | 0 | 150000 | 15 | 15 |
| 4a | 150000 | 19 | 19 | 9 | 1 | 150000 | 19 | 19 |
| 5a | 150000 | 53 | 53 | 31 | 14 | 150000 | 53 | 53 |
| 6a | 375000 | 79 | 79 | 198 | 2421 | 375000 | 80.76 | 80 |
| 7a | 375000 | 68.82 | 68 | 385 | 1619 | 375000 | 68.96 | 68 |
| 8a | 375000 | 144.92 | 144 | 366 | 6443 | 375000 | 148.08 | 147 |
| 9a | 750000 | 150.98 | 150 | 345 | 20401 | 750000 | 152.42 | 152 |
| 10a | 750000 | 218.44 | 218 | 442 | 14926 | 1500000 | 220.28 | 219 |

Table 5.1: These are the results for problem1a up to problem10a. 'ACO' gives the results obtained with the pure ACO approach and 'EP' gives Liang et Al.'s results. 'sol' indicates the total number of solutions the algorithms were maximally allowed to investigate, 'avg' indicates how many stocks were used on average, 'opt' indicates the number of stocks in the best solution, 'it' indicates after how many iterations this number of stocks was first found, and 'time' indicates the average running time for one problem in CPU seconds (until the theoretical optimum or the maximum number of iterations was reached).

| problem | sol | avg | opt | it | time |
|---:|---:|---:|---:|---:|---:|
| 1a | 1000 | 9.00 | 9 | 1.0 | 0 |
| 2a | 2000 | 23.00 | 23 | 1.0 | 0 |
| 3a | 10000 | 15.00 | 15 | 60.0 | 1 |
| 4a | 10000 | 19.00 | 19 | 1.0 | 0 |
| 5a | 10000 | 53.00 | 53 | 1.4 | 0 |
| 6a | 10000 | 79.00 | 79 | 16.3 | 79 |
| 7a | 10000 | 68.00 | 68 | 57.8 | 6 |
| 8a | 10000 | 144.30 | 144 | 225.3 | 312 |
| 9a | 20000 | 150.00 | 150 | 98.7 | 375 |
| 10a | 20000 | 217.66 | 217 | 397.4 | 2297 |

Table 5.2: These are the results obtained with the ACO algorithm with local search for problem1a up to problem10a. 'sol' again indicates how many solutions the ants could maximally build, 'avg' how many stocks were used on average, 'opt' the number of stocks in the best solution, 'it' after how many iterations this number of stocks was first found, and 'time' how many CPU seconds the program needed on average for one problem.

problem 10a. With t-tests it can be shown that for all four problems the results with the pure ACO algorithm are worse with a probability of 100%.

The local search also solves the problem of long running times. Especially for the bigger problems, enormous speed-ups have been realised, thereby countering one of the disadvantages of the ACO approach. There is however one problem for which the ACO with local search needed more time: problem 3a. Also in number of iterations, this problem seems to give the ACO with local search more difficulties: 60 iterations on average as opposed to 3. One aspect in which problem 3a differs from the others is that in the optimal solution, all stocks are fully used: there is no waste. I cannot, however, see why this should give problems for the ACO with local search in particular. Maybe the way the local search tries to find better optima is not good for this particular problem.

### 5.2.2 Tests for the BPP

In [Falkenauer, 1996], Falkenauer compares his HGGA to Martello and Toth's RA. He uses 8 different sets of 20 test problems, and runs the algorithm just once on each of these. The first four sets contain problems with a bin capacity of 150 and item sizes uniformly distributed between 20 and 100 (because this kind of problem appeared to be the hardest in Martello and Toth's work). He uses four different problem sizes: 120 items, 250, 500 and 1000. For each size, 20 different problems were created randomly. These problems are further referred to as the uniform problems. 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, two of which 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 into a bin, or two large ones, but then the optimum will inevitably be missed. Again, 20 different problems were created for four different 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.

I ran both ACO algorithms on each instance of every problem set, except for the fourth (uniform problems of size 1000). For those problems, I only ran the ACO with local search, because the computation times for the pure ACO algorithm were prohibitive. The maximum number of solutions for the pure ACO algorithm was defined as before, by multiplying the population size (always 100) and the number of generations (2000 for the two smallest uniform problem sets and 5000 for the largest, 1000 for the two smallest triplet sets and 2000 for the two largest). The ACO algorithm with local search was always allowed 10 times less solutions. The results for the RA were obtained by Falkenauer. He let it do at maximum 1500000 backtracks, except when that was finished faster (in CPU seconds) than the run of HGGA. In that case, extra backtracks were allowed. For both ACO algorithms, runs were done with $\beta$ values of 2, 5 and 10. The ACO algorithm with local search always needed a similar or lower $\beta$ than the pure ACO. In the tables below, only the results for the best $\beta$ values are reported.

The results for the uniform problems are reported in tables 5.3, 5.4, 5.5 and 5.6 below. From these results, it is clear that the pure ACO algorithm cannot beat HGGA. For the problems of size 120, it does almost equally well, and for the ones of size 250, it equals HGGA in 11 out of 20 problems. For the problems of size 500, however, it always does slightly worse than HGGA. RA also does slightly better than the pure ACO on the smallest problems, but for the larger ones, it does clearly worse. The disadvantage of the pure ACO algorithm pointed out above, namely that it is slow for big problems, can again be observed here.

The ACO with local search does clearly better than the pure ACO. Especially on the smallest problems (size 120), it does very well. For those problems, Falkenauer's HGGA finds the theoretical optimum in all but two cases (problem 9 and 20). For these two cases, Falkenauer conjectures that the optimum cannot be reached, as neither HGGA nor RA find it. It is therefore all the more remarkable that the hybridised ACO algorithm finds these optima. For the larger problems, the ACO with local search does not always find the solution found by HGGA, and for the largest problems (size 1000), it always does slightly worse than HGGA. It does, however, always beat RA and the pure ACO algorithm. Like for the CSP problems, the local search also gives an enormous speed-up, making it much faster than the other algorithms for the two smallest problem sets, and comparable to HGGA and RA for the larger ones.

| Run | HGGA | | RA | | ACO | | HACO | |
|---|---|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time | bins | time |
| 1 | 48 | 15 | 48 | 0 | 48 | 80 | 48 | 1 |
| 2 | 49 | 0 | 49 | 0 | 49 | 58 | 49 | 1 |
| 3 | 46 | 6 | 46 | 29 | 46 | 102 | 46 | 1 |
| 4 | 49 | 50 | 49 | 0 | 49 | 226 | 49 | 1 |
| 5 | 50 | 0 | 50 | 0 | 50 | 64 | 50 | 1 |
| 6 | 48 | 19 | 48 | 0 | 48 | 110 | 48 | 1 |
| 7 | 48 | 19 | 48 | 0 | 48 | 212 | 48 | 1 |
| 8 | 49 | 22 | 49 | 0 | 49 | 276 | 49 | 1 |
| 9 | 51 | 3669 | 51 | 3681 | 51 | 1696 | 50 | 3 |
| 10 | 46 | 40 | 46 | 0 | 47 | 1236 | 46 | 3 |
| 11 | 52 | 0 | 52 | 0 | 52 | 85 | 52 | 1 |
| 12 | 49 | 24 | 49 | 0 | 49 | 120 | 49 | 1 |
| 13 | 48 | 26 | 48 | 0 | 49 | 1269 | 48 | 1 |
| 14 | 49 | 0 | 49 | 0 | 49 | 51 | 49 | 1 |
| 15 | 50 | 0 | 50 | 0 | 50 | 79 | 50 | 1 |
| 16 | 48 | 11 | 48 | 0 | 48 | 92 | 48 | 1 |
| 17 | 52 | 0 | 52 | 0 | 52 | 102 | 52 | 1 |
| 18 | 52 | 76 | 52 | 0 | 52 | 158 | 52 | 1 |
| 19 | 49 | 14 | 49 | 0 | 49 | 108 | 49 | 1 |
| 20 | 50 | 3635 | 50 | 3679 | 50 | 1269 | 49 | 6 |
| Averages | | 381 | | 370 | | 370 | | 1 |

Table 5.3: These are the results for the uniform problems of size 120. 'bins' contains the number of bins in the best solution, and 'time' the running time in CPU seconds. 'HGGA' gives the results of Falkenauer's algorithm, 'RA' for Martello and Toth's algorithm, 'ACO' for the pure ACO approach, and HACO for the ACO algorithm hybridised with local search. The $\beta$ value used here was 2 for both the pure ACO and the ACO with local search.

| Run | HGGA | | RA | | ACO | | HACO | |
|---|---|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time | bins | time |
| 1 | 99 | 257 | 100 | 1002 | 100 | 3008 | 99 | 11 |
| 2 | 100 | 47 | 100 | 0 | 100 | 2485 | 100 | 1 |
| 3 | 102 | 224 | 102 | 0 | 102 | 916 | 102 | 3 |
| 4 | 100 | 27 | 100 | 0 | 100 | 962 | 100 | 1 |
| 5 | 101 | 164 | 101 | 4 | 102 | 3452 | 101 | 34 |
| 6 | 101 | 478 | 103 | 522 | 103 | 3407 | 102 | 308 |
| 7 | 102 | 15 | 102 | 0 | 102 | 349 | 102 | 1 |
| 8 | 104 | 6629 | 104 | 7412 | 104 | 3464 | 104 | 304 |
| 9 | 105 | 924 | 106 | 1049 | 106 | 3622 | 106 | 317 |
| 10 | 101 | 158 | 102 | 597 | 102 | 3371 | 101 | 3 |
| 11 | 105 | 96 | 106 | 377 | 105 | 1035 | 105 | 2 |
| 12 | 101 | 240 | 102 | 1076 | 102 | 3365 | 101 | 110 |
| 13 | 106 | 5997 | 106 | 6101 | 106 | 3592 | 106 | 292 |
| 14 | 103 | 6347 | 103 | 6969 | 103 | 3519 | 103 | 312 |
| 15 | 100 | 83 | 100 | 0 | 100 | 1549 | 100 | 1 |
| 16 | 105 | 4440 | 106 | 4673 | 106 | 3481 | 106 | 305 |
| 17 | 97 | 255 | 98 | 545 | 98 | 3102 | 97 | 7 |
| 18 | 100 | 39 | 100 | 0 | 100 | 487 | 100 | 1 |
| 19 | 100 | 247 | 100 | 0 | 101 | 3384 | 101 | 313 |
| 20 | 102 | 68 | 102 | 0 | 102 | 980 | 102 | 1 |
| Averages | | 1337 | | 1516 | | 2476 | | 116 |

Table 5.4: The results for the uniform problems of size 250. $\beta$ was 10 for both ACO and HACO.

| Run | HGGA | | RA | | ACO | | HACO | |
|---|---|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time | bins | time |
| 1 | 198 | 481 | 201 | 987 | 200 | 24271 | 199 | 957 |
| 2 | 201 | 178 | 202 | 869 | 202 | 26512 | 202 | 908 |
| 3 | 202 | 348 | 204 | 911 | 203 | 25753 | 202 | 772 |
| 4 | 204 | 11121 | 206 | 11412 | 206 | 25351 | 205 | 1143 |
| 5 | 206 | 268 | 209 | 844 | 207 | 25351 | 206 | 59 |
| 6 | 206 | 130 | 207 | 818 | 206 | 20750 | 206 | 13 |
| 7 | 207 | 1656 | 210 | 1854 | 209 | 26826 | 208 | 890 |
| 8 | 204 | 1835 | 207 | 2085 | 206 | 23968 | 205 | 897 |
| 9 | 196 | 502 | 198 | 1222 | 197 | 24331 | 197 | 929 |
| 10 | 202 | 93 | 204 | 962 | 203 | 24497 | 202 | 66 |
| 11 | 200 | 106 | 202 | 894 | 201 | 25878 | 200 | 3 |
| 12 | 200 | 152 | 202 | 793 | 202 | 24527 | 200 | 335 |
| 13 | 199 | 1019 | 202 | 1258 | 201 | 24904 | 200 | 936 |
| 14 | 196 | 136 | 197 | 860 | 197 | 25131 | 197 | 951 |
| 15 | 204 | 952 | 205 | 1203 | 205 | 25546 | 204 | 30 |
| 16 | 201 | 375 | 203 | 783 | 203 | 19153 | 201 | 26 |
| 17 | 202 | 163 | 204 | 733 | 203 | 18561 | 202 | 20 |
| 18 | 198 | 337 | 201 | 755 | 199 | 18072 | 198 | 606 |
| 19 | 202 | 144 | 205 | 638 | 203 | 19558 | 202 | 30 |
| 20 | 196 | 307 | 199 | 819 | 198 | 17950 | 197 | 963 |
| Averages | | 1015 | | 1535 | | 23378 | | 527 |

Table 5.5: The uniform problems of size 500. $\beta$ was 10 for ACO and 2 for HACO.

| Run | HGGA | | RA | | HACO | |
|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time |
| 1 | 399 | 2925 | 403 | 3279 | 400 | 3047 |
| 2 | 406 | 4040 | 410 | 4887 | 408 | 3017 |
| 3 | 411 | 6262 | 416 | 6606 | 412 | 3001 |
| 4 | 411 | 32714 | 416 | 40286 | 413 | 2961 |
| 5 | 397 | 11862 | 401 | 20690 | 399 | 3073 |
| 6 | 399 | 3774 | 402 | 4216 | 401 | 3027 |
| 7 | 395 | 3033 | 398 | 3450 | 396 | 3000 |
| 8 | 404 | 9879 | 406 | 12674 | 405 | 3028 |
| 9 | 399 | 5585 | 402 | 6874 | 401 | 3006 |
| 10 | 397 | 8126 | 402 | 9568 | 400 | 2924 |
| 11 | 400 | 3359 | 404 | 3543 | 401 | 3021 |
| 12 | 401 | 6782 | 404 | 7422 | 403 | 3073 |
| 13 | 393 | 2537 | 396 | 2714 | 394 | 3019 |
| 14 | 396 | 11829 | 401 | 23319 | 398 | 3111 |
| 15 | 394 | 5838 | 399 | 6771 | 396 | 3070 |
| 16 | 402 | 12611 | 407 | 20458 | 405 | 3032 |
| 17 | 404 | 2379 | 407 | 3139 | 405 | 3010 |
| 18 | 404 | 2379 | 407 | 2506 | 405 | 2941 |
| 19 | 399 | 1330 | 403 | 1353 | 401 | 3000 |
| 20 | 400 | 3564 | 405 | 4110 | 402 | 3024 |
| Averages | | 7059 | | 9393 | | 3019 |

Table 5.6: The uniform problems of size 1000. $\beta$ was 5 for HACO. The pure ACO was not run on these problems, as computation times were too long.

The tables 5.7, 5.8, 5.9 and 5.10 show the results for the triplet problems. The pure ACO algorithm again fails to beat HGGA. RA has some results for the smallest problems (size 60) which are better than the ACO's, but as the problem size increases, it becomes clear that RA does much worse than pure ACO. An interesting fact is that the ACO algorithm has a very constant performance: although it never finds the optimum for the smallest problems, it is always close to it. RA 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. It can again be observed that the ACO algorithm is quite slow for large problems.

The ACO algorithm with local search does not better than the pure ACO algorithm for the two smallest problem sets. For the two largest problem sets, it does beat pure ACO. For none of the triplet problems, it manages to equal HGGA's performances. The large speed-up compared to pure ACO can again be observed.

It is kind of disappointing to see that HGGA does better than both ACO algorithms on many problem instances. There might be an explanation why ACO does not do so well on these problems, though. The way ACO works is by reinforcing good solutions. Solution parts that appear in many good solutions will get a lot of pheromone. If, however, a large number of solution parts are equally likely to appear in good solutions, ACO fails to differentiate them and performs poorly (see also [Bonabeau et al., 1999, Bonabeau et al., 2000]). In other words, it is important that the problems have enough structure. For the QAP (see [Gambardella et al., 1999b]), it was observed that ACO did very well on real-life problems, but could not compete with the best algorithms when it came to randomly constructed artificial problems, because those lacked structure. The structure of a search space can be expressed in the fitness-distance correlation (FDC) ([Jones and Forrest, 1995]). This is the correlation between the fitness of a solution and its distance to the global optimum. So it expresses whether good solutions can guide the algorithm towards the global optimum. The BPP test problems used above were all constructed randomly. Especially the triplets (which the ACO has the most difficulties with) were created to be artificially hard. It would be interesting to calculate the FDC for these problems, to see if they possess the necessary structure to be easily solved with ACO. Also, it would be interesting to compare ACO with HGGA on other kinds

| Run | HGGA | | RA | | ACO | | HACO | |
|---|---|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time | bins | time |
| 1 | 20 | 4 | 20 | 10 | 21 | 292 | 21 | 37 |
| 2 | 20 | 6 | 20 | 13 | 21 | 406 | 21 | 42 |
| 3 | 20 | 2 | 23 | 564 | 21 | 294 | 21 | 35 |
| 4 | 20 | 6 | 22 | 445 | 21 | 286 | 21 | 38 |
| 5 | 20 | 1 | 22 | 405 | 21 | 293 | 21 | 35 |
| 6 | 20 | 9 | 22 | 415 | 21 | 317 | 21 | 38 |
| 7 | 20 | 284 | 22 | 486 | 21 | 325 | 21 | 38 |
| 8 | 21 | 295 | 22 | 396 | 21 | 292 | 21 | 37 |
| 9 | 20 | 7 | 22 | 452 | 21 | 290 | 21 | 38 |
| 10 | 20 | 6 | 20 | 10 | 21 | 308 | 21 | 39 |
| 11 | 20 | 15 | 20 | 1 | 21 | 307 | 21 | 40 |
| 12 | 20 | 1 | 20 | 6 | 21 | 267 | 21 | 34 |
| 13 | 20 | 3 | 20 | 2 | 21 | 373 | 21 | 40 |
| 14 | 20 | 5 | 22 | 385 | 21 | 302 | 21 | 36 |
| 15 | 20 | 6 | 22 | 401 | 21 | 293 | 21 | 38 |
| 16 | 20 | 3 | 23 | 537 | 21 | 332 | 21 | 38 |
| 17 | 20 | 2 | 23 | 528 | 21 | 276 | 21 | 34 |
| 18 | 20 | 9 | 22 | 430 | 21 | 283 | 21 | 37 |
| 19 | 21 | 281 | 22 | 386 | 21 | 293 | 21 | 41 |
| 20 | 20 | 2 | 22 | 400 | 21 | 303 | 21 | 40 |
| Averages | | 47 | | 313 | | 307 | | 38 |

Table 5.7: The triplets of size 60. $\beta$ was 2 for both ACO and HACO.

| Run | HGGA | | RA | | ACO | | HACO | |
|---|---|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time | bins | time |
| 1 | 40 | 121 | 44 | 844 | 41 | 1394 | 41 | 135 |
| 2 | 40 | 104 | 43 | 823 | 41 | 1367 | 41 | 131 |
| 3 | 40 | 96 | 43 | 956 | 41 | 1492 | 41 | 141 |
| 4 | 40 | 39 | 44 | 859 | 41 | 1400 | 41 | 188 |
| 5 | 40 | 76 | 45 | 1184 | 41 | 1505 | 41 | 151 |
| 6 | 40 | 149 | 45 | 1189 | 41 | 1451 | 41 | 153 |
| 7 | 40 | 47 | 45 | 1054 | 41 | 1398 | 41 | 165 |
| 8 | 40 | 61 | 43 | 777 | 41 | 1406 | 41 | 176 |
| 9 | 40 | 37 | 43 | 643 | 41 | 1403 | 41 | 184 |
| 10 | 40 | 256 | 44 | 1003 | 41 | 1381 | 41 | 138 |
| 11 | 40 | 103 | 44 | 886 | 41 | 1367 | 41 | 136 |
| 12 | 40 | 50 | 45 | 980 | 41 | 1403 | 41 | 182 |
| 13 | 40 | 43 | 44 | 1014 | 41 | 1313 | 41 | 136 |
| 14 | 40 | 57 | 44 | 835 | 41 | 1404 | 41 | 161 |
| 15 | 40 | 41 | 44 | 824 | 41 | 1366 | 41 | 134 |
| 16 | 40 | 47 | 44 | 873 | 41 | 1326 | 41 | 133 |
| 17 | 40 | 93 | 43 | 629 | 41 | 1414 | 41 | 169 |
| 18 | 40 | 51 | 44 | 790 | 41 | 1466 | 41 | 139 |
| 19 | 40 | 67 | 46 | 1171 | 41 | 1712 | 41 | 138 |
| 20 | 40 | 40 | 45 | 1076 | 41 | 1412 | 41 | 147 |
| Averages | | 79 | | 921 | | 1419 | | 152 |

Table 5.8: Triplets of size 120. $\beta$ was 5 for ACO and 2 for HACO.

| Run | HGGA | | RA | | ACO | | HACO | |
|---|---|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time | bins | time |
| 1 | 83 | 323 | 93 | 2381 | 85 | 14036 | 84 | 691 |
| 2 | 83 | 227 | 88 | 1526 | 85 | 11988 | 84 | 753 |
| 3 | 83 | 217 | 88 | 1455 | 85 | 11876 | 84 | 733 |
| 4 | 83 | 723 | 90 | 1717 | 85 | 12136 | 84 | 750 |
| 5 | 83 | 382 | 91 | 2513 | 85 | 11721 | 84 | 718 |
| 6 | 83 | 1717 | 90 | 2177 | 85 | 12513 | 84 | 743 |
| 7 | 83 | 1474 | 90 | 2108 | 85 | 11680 | 84 | 698 |
| 8 | 83 | 4400 | 92 | 2493 | 85 | 11626 | 84 | 723 |
| 9 | 83 | 615 | 91 | 2438 | 85 | 11956 | 84 | 726 |
| 10 | 83 | 318 | 90 | 1522 | 85 | 12148 | 84 | 709 |
| 11 | 83 | 777 | 94 | 2815 | 85 | 11974 | 84 | 720 |
| 12 | 83 | 191 | 90 | 1688 | 85 | 12053 | 84 | 707 |
| 13 | 83 | 262 | 89 | 1608 | 85 | 12218 | 84 | 711 |
| 14 | 83 | 360 | 91 | 2363 | 85 | 12519 | 84 | 700 |
| 15 | 83 | 204 | 89 | 1399 | 85 | 12792 | 84 | 728 |
| 16 | 83 | 76 | 91 | 2683 | 85 | 12748 | 84 | 721 |
| 17 | 83 | 667 | 90 | 2081 | 85 | 12798 | 84 | 739 |
| 18 | 83 | 307 | 90 | 2086 | 85 | 12655 | 84 | 737 |
| 19 | 83 | 294 | 91 | 2237 | 85 | 12086 | 84 | 727 |
| 20 | 83 | 1025 | 91 | 2199 | 85 | 11777 | 84 | 724 |
| Averages | | 728 | | 2074 | | 12265 | | 723 |

Table 5.9: Triplets of size 249. $\beta$ was 5 for ACO and 2 for HACO.

| Run | HGGA | | RA | | ACO | | HACO | |
|---|---|---|---|---|---|---|---|---|
| | bins | time | bins | time | bins | time | bins | time |
| 1 | 167 | 1807 | 184 | 5829 | 173 | 14905 | 170 | 2867 |
| 2 | 167 | 1582 | 181 | 3437 | 173 | 15216 | 171 | 2030 |
| 3 | 167 | 1235 | 177 | 2359 | 172 | 14946 | 171 | 2113 |
| 4 | 167 | 1822 | 180 | 3398 | 173 | 15949 | 171 | 2206 |
| 5 | 167 | 2355 | 181 | 3710 | 173 | 15782 | 170 | 4037 |
| 6 | 167 | 1424 | 183 | 10624 | 173 | 15650 | 171 | 2156 |
| 7 | 167 | 1161 | 183 | 5789 | 173 | 15729 | 170 | 2142 |
| 8 | 167 | 1504 | 183 | 5799 | 172 | 15439 | 170 | 2104 |
| 9 | 167 | 2138 | 177 | 2991 | 172 | 15628 | 171 | 2167 |
| 10 | 167 | 1550 | 185 | 5626 | 173 | 14921 | 170 | 2463 |
| 11 | 167 | 1053 | 179 | 3771 | 173 | 15275 | 171 | 2133 |
| 12 | 167 | 1335 | 178 | 3064 | 173 | 15538 | 171 | 2126 |
| 13 | 167 | 1502 | 187 | 5787 | 173 | 14428 | 170 | 2245 |
| 14 | 167 | 1951 | 181 | 4495 | 172 | 15851 | 173 | 2252 |
| 15 | 167 | 1474 | 183 | 5930 | 173 | 16325 | 170 | 4127 |
| 16 | 167 | 2351 | 181 | 5307 | 173 | 15729 | 170 | 2171 |
| 17 | 167 | 1179 | 183 | 5522 | 173 | 15244 | 171 | 2174 |
| 18 | 167 | 1754 | 183 | 6277 | 173 | 15228 | 172 | 2474 |
| 19 | 167 | 1776 | 180 | 4164 | 173 | 15303 | 170 | 2092 |
| 20 | 167 | 2307 | 188 | 6519 | 173 | 15223 | 173 | 2151 |
| Averages | | 1663 | | 5020 | | 15416 | | 2412 |

Table 5.10: Triplets of size 501. $\beta$ was 2 for both ACO and HACO.

of problems. The good performance of ACO with local search on the small uniform problems might suggest that there are certain problem classes for which ACO does better.

# Chapter 6

# Conclusions

In this dissertation, I have developed an ACO algorithm for the bin packing problem and the cutting stock problem. Artificial ants stochastically build new solutions, using a combination of 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. In this way, ideas of evolutionary algorithms and reinforcement learning are combined. The relative importance of the pheromone trail information as opposed to the heuristic information is defined by the parameter $\beta$, 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 in a preliminary optimisation phase.

Apart from a pure ACO approach, I also proposed a hybrid approach, which combines ACO with local search. The solutions constructed by the ants are taken to a local optimum by a search based on Martello and Toth's dominance criterion. This extended algorithm gave better and faster solutions than the pure ACO approach.

When compared to existing evolutionary approaches, both ACO algorithms managed to outperform Liang et Al.'s EP solution for the CSP and Martello and Toth's RA for the BPP. The pure ACO algorithm failed to compete with Falkenauer's HGGA, which is at the moment probably the best solution method for the BPP. The algorithm was also very slow for large problem instances. The hybridised ACO algorithm was much faster and could outperform HGGA on some small test problems, but gave equal or slightly worse results than HGGA on all of the other problems. Especially the artificially hard triplet problems seemed to pose more problems for the ACO algorithms than for HGGA. An advantage of ACO seems to be that it gives a steady performance: when it does not find the optimum, it always finds a near-optimal solution.

I think these are quite encouraging results. Especially the good performance of the hybrid algorithm on the small uniform problems, which could mean that there are certain kinds of problems where ACO is preferable over all other approaches. We should not forget that this is the first ACO application for these problem classes, and there is quite a lot of scope for more research to try and improve the current algorithm. It could for example be interesting to try out a different ACO approach. The ACO algorithm in this work was mainly based on MMAS, and it could be useful to implement features of the equally well performing ACS (see section 3.3). A second way to improve the algorithm could be by changing the local search algorithm. Alvim et Al.'s local search method (described in section 4.7) gives good results when used on its own, and it could be interesting to try to combine it with ACO. Thirdly, it could be interesting to try to change the value for $\beta$ dynamically. As for GA's (see [Mitchell, 1996]), it could well be that for some parameters the optimal value changes over the course of a single run. Maybe by adapting $\beta$ during the run, it could be possible to define it more generally and avoid the preliminary optimisation phase. Finally, it could be interesting to move away from a constructive ACO approach, and use the pheromone trail to change existing solutions, rather than to build new ones (like in HAS-QAP, described in section 3.4). Maybe it could even be used in combination with Falkenauer's HGGA replacing its mutation operator. In this way it might be possible to combine the best of both worlds.

Apart from making improvements to the algorithm developed in this dissertation, it could also be interesting to try to extend its application area. As mentioned in section 2.1, there are many variations possible to the traditional one-dimensional BPP and CSP. I don't think it would be very difficult to adapt the algorithm to take extra constraints into account: this is probably just a matter of limiting the items to choose from while building a solution. Taking extra goals, like contiguity, into account could be more tricky. One might be tempted to think that it suffices to adapt the fitness function, but this is probably not enough: the pheromone trail only contains information about the combination of item sizes in bins, and not about things like the order of the bins, which is important for contiguity. Also extending the algorithm for solving multiple stock length problems might be more difficult than it seems. You would need a three-dimensional pheromone trail to be able to express for which stock length certain combinations are good, and a mechanism to decide on the next stock length when building solutions. Finally, to solve multi-dimensional cutting and packing problems, you would need a completely different algorithm. It might still be possible to use ACO, but the implementation of it would probably be completely different from the one developed in this work.

# Appendix A

# The test problems for the CSP

These are the 10 single stock length test problems from [Liang et al., 2001].

**Problem1a**:

stock length: 14

items: 20

| Item length | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|
| No. Required | 5 | 2 | 1 | 2 | 4 | 2 | 1 | 3 |

**Problem2a**:

stock length: 15

items: 50

| Item length | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|
| No. Required | 4 | 8 | 5 | 7 | 8 | 5 | 5 | 8 |

**Problem3a**:

stock length: 25

items: 60

| Item length | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|
| No. Required | 6 | 12 | 6 | 5 | 15 | 6 | 4 | 6 |

**Problem4a**:

stock length: 25

items: 60

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item length | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| No. Required | 7 | 12 | 15 | 7 | 4 | 6 | 8 | 1 |

**Problem5a**:

stock length: 4300

items: 126

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item length | 1050 | 1100 | 1150 | 1200 | 1250 | 1300 | 1350 | 1650 |
| No. Required | 3 | 8 | 4 | 10 | 6 | 3 | 9 | 2 |
| Item length | 1700 | 1850 | 1900 | 1950 | 2000 | 2050 | 2100 | 2200 |
| No. Required | 5 | 13 | 15 | 6 | 11 | 6 | 15 | 4 |
| Item length | 2250 | 2350 | | | | | | |
| No. Required | 4 | 2 | | | | | | |

**Problem6a**:

stock length: 86

items: 200

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item length | 21 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| No. Required | 10 | 14 | 10 | 7 | 14 | 4 | 13 | 9 |
| Item length | 31 | 33 | 34 | 35 | 37 | 38 | 41 | 42 |
| No. Required | 5 | 10 | 13 | 10 | 11 | 15 | 12 | 15 |
| Item length | 44 | 47 | | | | | | |
| No. Required | 15 | 13 | | | | | | |

**Problem7a**:

stock length: 120

items: 200

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item length | 22 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| No. Required | 6 | 3 | 14 | 12 | 9 | 15 | 11 | 10 |
| Item length | 34 | 36 | 37 | 38 | 39 | 46 | 47 | 48 |
| No. Required | 11 | 13 | 4 | 3 | 6 | 14 | 7 | 3 |
| Item length | 52 | 53 | 54 | 56 | 58 | 60 | 63 | 64 |
| No. Required | 14 | 9 | 7 | 3 | 5 | 14 | 4 | 3 |

**Problem8a**:

stock length: 120

items: 400

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item length | 22 | 23 | 24 | 26 | 27 | 28 | 29 | 30 |
| No. Required | 12 | 8 | 27 | 15 | 25 | 7 | 10 | 22 |
| Item length | 31 | 36 | 39 | 41 | 42 | 48 | 49 | 50 |
| No. Required | 5 | 16 | 19 | 21 | 26 | 16 | 12 | 26 |
| Item length | 51 | 54 | 55 | 56 | 59 | 60 | 66 | 67 |
| No. Required | 20 | 25 | 9 | 17 | 22 | 14 | 17 | 9 |

**Problem9a**:

stock length: 120

items: 400

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item length | 21 | 22 | 24 | 25 | 27 | 29 | 30 | 31 |
| No. Required | 13 | 15 | 7 | 5 | 9 | 9 | 3 | 15 |
| Item length | 32 | 33 | 34 | 35 | 38 | 39 | 42 | 44 |
| No. Required | 18 | 17 | 4 | 17 | 20 | 9 | 4 | 19 |
| Item length | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 |
| No. Required | 9 | 12 | 15 | 3 | 20 | 14 | 15 | 6 |
| Item length | 53 | 54 | 55 | 56 | 57 | 59 | 60 | 61 |
| No. Required | 4 | 7 | 5 | 19 | 19 | 6 | 3 | 7 |
| Item length | 63 | 65 | 66 | 67 | | | | |
| No. Required | 20 | 5 | 10 | 17 | | | | |

**Problem10a**:

stock length: 120

items: 600

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Item length | 21 | 22 | 23 | 24 | 25 | 27 | 28 | 29 |
| No. Required | 13 | 19 | 24 | 20 | 23 | 24 | 15 | 5 |
| Item length | 30 | 31 | 33 | 35 | 36 | 39 | 40 | 41 |
| No. Required | 24 | 16 | 12 | 24 | 16 | 4 | 20 | 24 |
| Item length | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 50 |
| No. Required | 6 | 14 | 21 | 20 | 24 | 2 | 11 | 26 |
| Item length | 51 | 54 | 56 | 57 | 58 | 61 | 62 | 63 |
| No. Required | 23 | 25 | 8 | 16 | 10 | 14 | 6 | 19 |
| Item length | 64 | 65 | 66 | 67 | | | | |
| No. Required | 18 | 11 | 27 | 16 | | | | |

# Appendix B

# Pseudo code for the algorithms

In this appendix, an overview of both the pure ACO algorithm and the algorithm with local search is given in pseudo-code. The full source code is available on-line at http://www.aiai.ed.ac.uk/ johnl/antbin.

## B.1 The pure ACO algorithm

Initialise all pheromone trail entries to $\tau(0)$
**Loop** until the maximum number of iterations or the theoretical optimum is reached
    **Loop** for every ant
        **Loop** until all items are placed
            Open a new bin
            **Loop** until no remaining item fits in the bin anymore
                **Loop** for each remaining item size $j$ that still fits
                    Sum the pheromone between $j$ and all item sizes $i$
                        already in the bin
                    Divide this by the number of items in the bin
                    Calculate the probability for $j$ according to (4.6)
                **End Loop**
                Choose an item according to the calculated probabilities
            **End Loop**

**End Loop**
Calculate the fitness according to (4.2)
**End Loop**
Find the iteration best ant
Replace the globally best ant if the iteration best was fitter
**If** ((iteration number *mod* $\gamma$) = 0)
Use the globally best ant for pheromone updating
**Else**
Use the iteration best ant for pheromone updating
**End If**
Decrease all pheromone entries multiplying them with $\rho$ (with lower limit $\tau_{min}$)
**Loop** for every bin in the best ant's solution
**Loop** for every combination of item sizes $i$ and $j$ in the bin:
Increase $\tau(i, j)$ with the fitness of the solution
**End Loop**
**End Loop**
**End Loop**

## B.2 The ACO algorithm with local search

Initialise all pheromone trail entries to 0
**Loop** until the maximum number of iterations or the theoretical optimum is reached
**Loop** for every ant
**Loop** until all items are placed
Open a new bin
**Loop** until no remaining item fits in the bin anymore
**Loop** for each remaining item size $j$ that still fits
Sum the pheromone between $j$ and all item sizes $i$ already in the bin
Divide this by the number of items in the bin

```
                Calculate the probability for j according to (4.6)
            End Loop
            Choose an item according to the calculated probabilities
        End Loop
    End Loop
    Find a fixed number of the smallest bins
    Remove them from the solution and label their items as free items
    Loop for every one of the remaining bins
        Try to replace 2 current items by 2 free items, making the bin fuller
        Try to replace 2 current items by 1 free item
        Try to replace 1 current item by 1 free item
    End Loop
    Sort the remaining free items in non-ascending order of size
    Loop for every free item
        Place the item in the first bin of the current solution it fits in
        If no bin can can take the item
          Add a new bin with the item to the solution
        End If
    End Loop
    Calculate the ant's solution fitness according to (4.2)
  End Loop
  Find the iteration best ant
  Replace the globally best ant if the iteration best was fitter
  Decrease the pheromone on all edges multiplying it with ρ
  Loop for every bin in the globally best ant's solution
    Loop for every combination of item lengths i and j in the bin:
        Increase τ(i, j) with the fitness of the solution
    End Loop
  End Loop
End Loop
```

# Appendix C

# Published material

The following paper was written for the UK Workshop on Computational Intelligence, held on 10-12 September 2001 at the University of Edinburgh.

# Bibliography


[Alvim et al., 1999] Alvim, A. C. F., Glover, F. S., Ribeiro, C. C., and Aloise, D. J. (1999). Local search for the bin packing problem. http://citeseer.nj.nec.com/alvim99local.html.

[Baluja and Caruana, 1995] Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In *Proceedings of the Twelfth International Conference on Machine Learning, ML-95*, pages 38–46, Palo Alto, CA, USA. Morgan Kaufmann.

[Bauer et al., 1999] Bauer, A., Bullnheimer, B., Hartl, R. F., and Strauss, C. (1999). An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1445–1450, Piscataway, NJ, USA. IEEE Press.

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

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

[Bonabeau et al., 1999] Bonabeau, E., Dorigo, M., and Theraulez, G. (1999). *Swarm Intelligence: from natural to artificial intelligence*. Oxford University Press, Inc., New York, NY, USA.

[Bonabeau et al., 2000] Bonabeau, E., Dorigo, M., and Theraulez, G. (2000). Inspiration for optimization from social insect behaviour. *Nature*, 406:39–42.

[Bullnheimer et al., 1999] Bullnheimer, B., Hartl, R. F., and Strauss, C. (1999). A new rank based version of the ant system: A computational study. *Central European Journal For Operations Research and Economics*, 7(1):25–38.

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

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

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

[Dorigo, 1992] Dorigo, M. (1992). *Optimization, learning and natural algorithms*. PhD thesis, DEI, Polytecnico di Milano, Milan, Italy. In Italian.

[Dorigo and Caro, 1999] Dorigo, M. and Caro, G. D. (1999). The ant colony optimization meta-heuristic. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK.

[Dorigo et al., 1999] Dorigo, M., Caro, G. D., and Gambardella, L. M. (1999). Ant algorithms for discrete optimisation. *Artificial Life*, 5(2):137–172.

[Dorigo and Gambardella, 1996] Dorigo, M. and Gambardella, L. M. (1996). A study of some properties of ant-q. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 656–665, Berlin, Germany. Springer.

[Dorigo and Gambardella, 1997a] Dorigo, M. and Gambardella, L. M. (1997a). Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81.

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

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics B*, 26(1):29–41.

[Dorigo and Stützle, 2001] Dorigo, M. and Stützle, T. (2001). The ant colony optimization metaheuristic: Algorithms, applications, and advances. to appear in Handbook of Metaheuristics, F. Glover and G. Kochenberger.

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

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

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

[Gambardella et al., 1999a] Gambardella, L. M., Taillard, E. D., and Agazzi, G. (1999a). *MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows*, pages 63–76. McGraw Hill, London, UK.

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

[Gilmore and Gomory, 1961] Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting stock problem. *Operations Research*, 9:848–859.

[Haessler and Sweeney, 1991] Haessler, R. W. and Sweeney, P. E. (1991). Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150.

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

[Jones and Forrest, 1995] Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th international Conference on Genetic Algorithms*, pages 184–192, San Fransisco, CA, USA. Morgan Kaufmann.

[Leguizamon and Michalewicz, 1999] Leguizamon, G. and Michalewicz, Z. (1999). A new version of ant system for subset problems. In *Proceedings of the 1999 Congress of Evolutionary Computation*, pages 1459–1464, Piscataway, NJ, USA. IEEE Press.

[Liang et al., 2001] Liang, K.-H., Yao, X., Newton, C., and Hoffman, D. (2001). A new evolutionary approach to cutting stock problems with and without contiguity. To appear in Computers and Operations Research.

[Maniezzo et al., 1994] Maniezzo, V., Colorni, A., and Dorigo, M. (1994). The ant system applied to the quadratic assignment problem. Technical report, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

[Martello and Toth, 1990] Martello, S. and Toth, P. (1990). *Knapsack Problems, Algorithms and Computer Implementations*. John Wiley and Sons Ltd., England.

[Michel and Middendorf, 1999] Michel, R. and Middendorf, M. (1999). *An ACO algorithm for the shortest supersequence problem*, pages 51–61. McGraw Hill, London, UK.

[Mitchell, 1996] Mitchell, M. (1996). *An introduction to genetic algorithms*. The MIT Press, Cambridge, MA, USA.

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

[Stützle, 1998] Stützle, T. (1998). 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. Verlag Mainz.

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

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

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

[Winston, 1993] Winston, W. L. (1993). *Operations Research: Applications and Algorithms*. International Thompson Publishing, Belmont, CA, USA.