Journal of the Operational Research Society (2004) 55, 705–716

www.palgrave-journals.com/jors

Ant colony optimization and local search for bin packing and cutting stock problems

J Levine* and F Ducatelle

University of Edinburgh, Edinburgh, UK

The Bin Packing Problem and the Cutting Stock Problem are two related classes of NP-hard combinatorial optimization problems. Exact solution methods can only be used for very small instances, so for real-world problems, we have to rely on heuristic methods. In recent years, researchers have started to apply evolutionary approaches to these problems, including Genetic Algorithms and Evolutionary Programming. In the work presented here, we used an ant colony optimization (ACO) approach to solve both Bin Packing and Cutting Stock Problems. We present a pure ACO approach, as well as an ACO approach augmented with a simple but very effective local search algorithm. It is shown that the pure ACO approach can compete with existing evolutionary methods, whereas the hybrid approach can outperform the best-known hybrid evolutionary solution methods for certain problem classes. The hybrid ACO approach is also shown to require different parameter values from the pure ACO approach and to give a more robust performance across different problems with a single set of parameter values. The local search algorithm is also run with random restarts and shown to perform significantly worse than when combined with ACO.
Journal of the Operational Research Society (2004) **55,** 705–716. doi:10.1057/palgrave.jors.2601771

Keywords: ant colony optimization; bin packing; cutting stock

Introduction

The Bin Packing Problem (BPP) and the Cutting Stock Problem (CSP) are two classes of well-known NP-hard combinatorial optimization problems.[1] In the BPP, the aim is to combine items into bins of a certain capacity so as to minimize the total number of bins, whereas in the CSP, the aim is to cut items from stocks of a certain length, minimizing 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[1] and the reduction algorithm of Martello and Toth.[2] CSP instances are traditionally solved with sequential heuristics or methods based on linear programming.[3] 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[4–7] and evolutionary programming.[8] The most successful of these new approaches is Falkenauer's hybrid grouping genetic algorithm,[4] which combines a grouping based genetic algorithm with a simple local search inspired by Martello and Toth's work.

In this article, we propose an Ant Colony Optimization (ACO) approach to the BPP and the CSP. ACO is a new meta-heuristic for combinatorial optimization and other problems. The first ACO algorithm was developed by Dorigo, Maniezzo and Colorni in 1991, and published under the name Ant System (AS).[9] It was an application for the Travelling Salesman Problem (TSP), 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.[10,11] 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.[12]

Apart from a pure ACO approach, we also develop a hybrid ACO algorithm. This approach combines the ACO meta-heuristic with a simple but effective local search algorithm based on the Dominance Criterion of Martello and Toth.[2] Each ant's solution is improved by moving some of the items around, and the improved solutions are used to

*Correspondence: J Levine, Centre for Intelligent Systems and their Applications, School of Informatics, University of Edinburgh, Appleton Tower, Crichton Street, Edinburgh EH8 9LE, UK.
E-mail: johnl@inf.ed.ac.uk

update the pheromone trail. The reason for trying such an approach is the knowledge that ACO and local search can work as a complementary partnership.[11] ACO performs a rather coarse-grained search, providing good starting points for local search to refine the results.

This article is organized as follows. The next section introduces the two combinatorial optimization problems, and describes the most important existing solution methods for them. We then give a general introduction to ACO algorithms, describing AS and some of its extensions and applications. The following section contains a detailed explanation of how we applied ACO to the BPP and the CSP, and how the approach was augmented with local search. We then give our experimental results: 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. We also present results using local search from random initial positions, to see how much useful work the ACO is performing. The final section concludes with a summary of the work and an overview of possible future work on this subject.

Packing bins and cutting stock

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.

Both the BPP and the CSP belong to the large group of cutting and packing problems. Dyckhoff describes a common logical structure for this group of problems.[1] 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.

When classifying the BPP and the CSP according to his typology, Dyckhoff only makes a distinction between them based on the one criterion, namely 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). While this means that the difference between the two problem types is a rather subjective and gradual one, it has still been important enough to dictate totally different solution approaches for the two problems, as outlined in the Introduction.

Bischoff and Wäscher[13] 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[1] 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 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 research. 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.

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. Another often used fast heuristic is best fit decreasing (BFD). The only difference from FFD is that the items are not placed in the first bin that can hold them, but in the best-filled bin that can hold them. This makes the algorithm slightly more complicated, but surprisingly enough, no better. Both heuristics have a guaranteed worst-case performance of $\frac{11}{9}Opt + 4$, in which Opt is the number of bins in the optimal solution to the problem.[14]

Apart from these fast algorithms, the BPP can also be solved with Martello and Toth's reduction procedure (MTP).[2] This is slower (certainly for bigger problems), but gives excellent results. The basis of the MTP is the notion of dominating bins: when you have two bins B_1 and B_2, and there is a subset $\{i_1,\ldots,i_l\}$ of B_1 and a partition $\{P_1,\ldots,P_l\}$ 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. This means that a solution which contains B_1 will not have more bins than a solution containing B_2. The MTP tries to find bins that dominate all other bins. When such a bin is found, the problem is reduced by removing the items of the dominating bin. In order to avoid that the algorithm runs into exponential time, only dominating bins of maximum three items are considered.

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.[15] This method is based on the LP-relaxation of the problem:

$$\begin{array}{ll} \text{Minimize} & \sum_j X_j \\ \text{Subject to} & \sum_j A_{ij} X_j \geqslant R_i \quad \text{for all } i \\ & X_j \geqslant 0 \end{array} \qquad (1)$$

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)). The goal of this model is to fill the stock length with items while maximizing the total benefit this will give to the LP model (indicated by the shadow prices U_i).

$$\begin{array}{ll} \text{Maximize} & \sum_i U_i A_i \\ \text{Subject to} & \sum_i L_i A_i \leqslant L \\ & A_i \geqslant 0 \quad A_i \text{ is an integer} \end{array} \qquad (2)$$

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 (1). More details about this can be found in Haessler and Sweeney[3] and Winston.[16]

An alternative for these LP-based solution methods are the sequential heuristic procedures (SHP). They construct a solution by making one pattern at a time until all order requirements are satisfied. When making a pattern, other goals than waste minimization 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.[3]

Evolutionary approaches

In recent years, people have tried various sorts of evolutionary approaches for the BPP and the CSP.[5–7,17] This section gives a short introduction to Falkenauer's hybrid grouping genetic algorithm (HGGA) for the BPP[4] and the Liang *et al*[8] evolutionary programming approach (EP) for the CSP, because these are the algorithms the ACO approach of this project is compared to in our experimental work.

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

Compared to HGGA, Liang *et al*'s EP for the CSP is a very simple algorithm. The EP uses an order-based approach for representing the solutions, but without crossover. This is motivated by the fact that Hinterding and Khan[5] found that the performance of an order-based GA for the CSP was degraded when crossover was used. Mutation in Liang *et al*'s EP happens by swapping elements around: every parent produces one child by swapping three elements around twice. 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 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.

Ant colony optimization

ACO is a multi-agent meta-heuristic for combinatorial optimization and other problems. It is inspired by the capability of real ants to find the shortest path between their nest and a food source. The first ACO algorithm, AS, was an application to solve the TSP, developed in 1992 by Dorigo. AS became very popular after its publication in 1996.[9] Many researchers have since developed improvements to the original algorithm, and have applied them to a range of different problems.

Ant system

ACO algorithms were originally inspired by the ability of real ants to find the shortest path between their nest and a food source. The key to this ability lies in the fact that ants leave a pheromone trail behind while walking.[10] Other ants can smell this pheromone, and follow it. When a colony of ants is presented with two possible paths, each ant initially chooses one randomly, resulting in 50% going over each path. It is clear, however, that the ants using the shortest path will be back faster. So, immediately after their return there will be more pheromone on the shortest path, influencing other ants to follow this path. After some time, this results in the whole colony following the shortest path.

AS is an ACO algorithm based on this biological metaphor which was first applied to the TSP. It associates an amount of pheromone $\tau(i, j)$ with the connection between two cities i and j. Each ant is placed on a random start city, and builds a solution going from city to city, until it has visited all of them. The probability that an ant k in a city i chooses to go to a city j next is given by

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

In this equation, $\tau(i, j)$ is the pheromone between i and j and $\eta(i, j)$ is a simple heuristic guiding the ant. The value of the heuristic is the inverse of the cost of the connection between i and j. So the preference of ant k in city i for city j is partly defined by the pheromone between i and j, and partly by the heuristic favourability of j after i. It is the parameter β which defines the relative importance of the heuristic information as opposed to the pheromone information. $J_k(i)$ is the set of cities that have not yet been visited by ant k in city i.

Once all ants have built a tour, the pheromone is updated. This is done according to these equations:

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

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

Equation (4) consists of two parts. The left part makes the pheromone on all edges decay. The speed of this decay is defined by ρ, the evaporation parameter. The right part increases the pheromone on all the edges visited by ants. The amount of pheromone an ant k deposits on an edge is defined by L_k, the length of the tour created by that ant. In this way, the increase of pheromone for an edge depends on the number of ants that use this edge, and on the quality of the solutions found by those ants.

Extensions and other applications

AS performed well on relatively small instances of the TSP, but could not compete with other solution approaches on larger instances. Later, many improvements to the original AS have been proposed, which also performed well on the bigger problems. Examples of these improvements are Ant Colony System[18] and MAX-MIN Ant System.[19] Also, ACO solutions have been developed for many other combinatorial optimization problems. Algorithms have been proposed for the quadratic assignment problem,[20,21] scheduling problems,[22,23] the vehicle routing problem (VRP),[24] the graph colouring problem,[25] the shortest common super-sequence problem,[26] the multiple knapsack problem,[27] and many others.

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.[12] He uses ACO to combine a GA and a many-agent search model (MA) into one-hybrid algorithm: 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. Although Bilchev's article is not very clear about implementation details, the results do suggest that a model in which well-defined heuristics cooperate can outperform any of its composing algorithms.

Applying ACO to the BPP and CSP

This section describes how the ACO meta-heuristic was adapted to solve the BPP and the CSP. The first section explains how the pheromone trail was defined, the second describes which heuristic was used, the third section gives details about how the ants build solutions, the fourth shows how the pheromone trail is updated, and the fifth section talks about the fitness function that was used to guide the algorithm towards better solutions. After that, the final section explains how the local search was added to improve the performance of the algorithm.

The pheromone trail definition

The quality of an ACO application depends very much on the definition of the meaning of the pheromone trail.[11] It is crucial to choose a definition conform the nature of the problem. The BPP and the CSP are grouping problems. What you essentially want to do is split the items into groups. This is in contrast to the TSP and most other problems ACO has been applied to. The TSP is an ordering problem: the aim is to put the different cities in a certain order. This is translated in the meaning of the pheromone

trail for the TSP: it encodes the favourability of visiting a certain city j after another city i.

Costa and Hertz[25] also use ACO to solve a grouping problem, namely the Graph Colouring Problem (GCP). In the GCP, a set of nodes is given, with undirected edges between them. The aim is to colour the nodes in such a way that no nodes of the same colour are connected. So, in fact, you want to group the nodes into colours. Costa and Hertz use a grouping-based approach, in which the pheromone trail between node i and node j encodes the favourability of having these nodes in the same colour. The pheromone matrix is of course symmetric ($\tau(i, j) = \tau(j, i)$).

We will define our pheromone trail in a similar way to Costa and Hertz: $\tau(i, j)$ encodes the favourability of having an item of size i and size j in the same bin (or stock). There is of course one important difference between the GCP on the one hand, 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 can be several items of size i and size j. Intuitively, this seems to give our approach two important advantages. Firstly, the pheromone matrix is very compact, especially for the CSP, where the number of different item sizes is few compared to the number of actual items. Secondly, the pheromone matrix encodes good packing patterns by reinforcing associations between sizes: once a good pattern has been found, then it can be used repeatedly when solving the problem.

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. One of the simplest and most effective heuristic methods for solving the CSP and BPP is first-fit decreasing: the items are sorted into order of size and then, starting with the largest, placed into the first bin that they still fit in.

However, since we will be filling the bins one by one, instead of placing the items one by one, we have to reformulate FFD slightly. Starting with a set of empty bins, we will fill each bin in turn by repeatedly placing the largest item from those remaining which will still fit into the bin. If no items left are small enough to fit into the bin, a new bin is started.

This procedure results in the FFD solution, but is more useful for our purposes: it shows us that the heuristic favourability of an item is directly related to its size. In other words, when choosing the next item to pack into the current bin, large items should be favoured over small items. This can be achieved by setting the heuristic function for an item being considered to be equal to its size; in other words $\eta(j) = j$.

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

$$p_k(s,b,j) = \begin{cases} \frac{[\tau_b(j)].[\eta(j)]^\beta}{\sum_{g \in J_k(s,b)} [\tau_b(g)].[\eta(g)]^\beta} & \text{if } j \in J_k(s,b) \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

In this equation, $J_k(s, b)$ is the set of items that qualify for inclusion in the current bin. They are the items that are still left after partial solution s is formed, and are light enough to fit in bin b. $\eta(j)$ is the weight of item j. The pheromone value $\tau_b(j)$ for an item j in a bin b is given in Equation (7). It is the sum of all the pheromone values between item j and the items i that are already in bin b, divided by the number of items in b. If b is empty, $\tau_b(j)$ is set to 1. This approach is similar to the one followed by Costa and Hertz.

$$\tau_b(j) = \begin{cases} \frac{\sum_{i \in b} \tau(i,j)}{|b|} & \text{if } b \neq \{\} \\ 1 & \text{otherwise} \end{cases} \quad (7)$$

Updating the pheromone trail

For the updating of the pheromone trail, we mainly followed the approach of Stützle and Hoos's MAX–MIN Ant System (MMAS).[19] We chose this version of the ACO algorithm because it is simple to understand and implement, and in the same time gives very good performance.

In MMAS, only the best ant is allowed to place pheromone after each iteration. We adapted Equation (4) to reflect this. The equation is further changed because of the nature of the BPP and the CSP: as mentioned before, item sizes i and j are not unique, and they might go together several times in the bins of the best solution. We will increase $\tau(i, j)$ for every time i and j are combined. So finally, we get Equation (8). In this equation, $t(i, j)$ indicates how many times i and j go together in the best solution s^{best}.

$$\tau(i,j) = \rho.\tau(i,j) + t(i,j).f(s^{\text{best}}) \quad (8)$$

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

Another way of enhancing exploration is obtained by defining an upper and lower limit (τ_{max} and τ_{min}) for the pheromone values (hence the name MAX–MIN). Stützle and Hoos define the value for the upper and lower limit algebraically. In our approach, we cannot use an upper limit. This is because, depending on how many times item sizes appear together in the good solutions (so m in Equation (8)), pheromone values get reinforced more, and they evolve to different values. We would have to use different maximum values for different entries in the pheromone matrix.

We do use the lower limit τ_{min}, though. Stützle and Hoos calculate the value for τ_{min} based on *pbest*, the probability of constructing the best solution found when all the pheromone values have converged to either τ_{max} or τ_{min}. An ant constructs the best solution found if it adds at every point during solution construction the item with the highest pheromone value. Starting from this, Stützle and Hoos find the following formula for τ_{min}[19]:

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

In this equation n is 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 $n/2$. In our approach, we used this formula, but replaced τ_{max} in it by $1/(1-\rho)$. This is in fact an approximation of τ_{max} as calculated by Stützle and Hoos for values for m of 0 or 1, replacing the fitness of the best solution by 1 (for most problems, the fitness value of the optimal solution using the fitness function given in the next section lies somewhere between 0.95 and 1). The result is Equation (10). The fact that several combinations of the same items are possible interferes quite severely with the calculations to get to Equation (9), and the value of *pbest* should therefore only be seen as a crude approximation of the real probability to construct the best solution.

$$\tau_{min} = \frac{1/(1 - \rho)(1 - \sqrt[n]{pbest})}{(avg - 1)\sqrt[n]{pbest}} \tag{10}$$

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

The fitness function

In order to guide the algorithm towards good solutions, we need to be able to assess the quality of the solutions. So we need a fitness function. A straightforward choice would be to take the inverse of the number of bins. As Falkenauer[4] 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 all these get the same fitness value, there is no way they can guide the algorithm towards an optimum, and the problem becomes a needle-in-a-haystack.

So, instead, we chose to use the function proposed by Falkenauer and Delchambre[17] to define the fitness of a solution s:

$$f(s) = \frac{\sum_{i=1}^{N} (F_i/C)^k}{N} \tag{11}$$

In this equation, N is the number of bins (stocks), F_i the total contents of bin i, and C the maximum contents of a bin. k is the parameter that defines how much stress we put on the nominator of the formula (the filling of the bins) as opposed to the denominator (the total number of bins). Setting k to 1 comes down to using the inverse of the number of bins. By increasing k, we give a higher fitness to solutions that contain a mix of well-filled and less well-filled bins, rather than equally filled bins. Falkenauer and Delchambre report that a value of 2 seems to be optimal. Values of more than 2 can lead to premature convergence, as the fitness of suboptimal solutions can come too close to the fitness of optimal solutions. Falkenauer[4] proves algebraically that if k is set to a value large 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.

Adding local search

It is known that the performance of ACO algorithms can sometimes be greatly improved when coupled to local search algorithms.[11] 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*.[28] 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 good results. However, for combination with an ACO algorithm, a reasonably fast and simple procedure was needed. Therefore, a local search procedure based on the local optimization algorithm used in Falkenauer's HGGA (as described in the section on evolutionary bin-packing algorithms) 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 the HGGA version of the local search algorithm, a number of items of the initial solution are made free. The algorithm then 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 (as described in the section on the bin packing problem), 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 the ACO algorithm, every solution created by an ant is taken through a local optimization phase. In this phase, the n least-filled bins are destroyed, and their items become free (the number of bins to be destroyed is defined empirically, see below). 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, to create a complete new solution. The complete local search procedure is then repeated with this new solution: the n least-filled bins are emptied and their items redistributed. This procedure is iterated until no improvement in fitness is detected between the solutions before and after the local search is applied. Hence, the local search is in fact a hill-climbing algorithm which takes the original solution created by the ACO procedure to the nearest local optimum.

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: |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 → 3 5 2 new free: 4, 3, 3, 3
Second bin: 6 2 1 → 6 4 new free: 3, 3, 3, 2, 1
Third bin: 7 2 → 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 → 5 4 1
Make new bin: 3 3 2 2
Final solution: |3 5 2 | 6 4 | 7 3 | 5 4 1| 3 3 2 2 |

Repeat the procedure: no further improvement possible

Figure 1 An example of the use of the local search algorithm.

A complete example of the local search phase is given in Figure 1. The pheromone then is updated using the locally improved solutions.

Experimental results

This section describes the results of our experiments. First, the different parameter values are defined, and then the pure ACO algorithm is compared to Liang *et al*'s Evolutionary Programming approach,[8] Martello and Toth's Reduction Algorithm[2] and Falkenauer's HGGA.[4] We then present the results for the hybrid ACO approach with local search on the same problems. Finally, we show the results for the local search procedure alone, using random restarts rather than ACO to create the initial solutions.

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 our tests to define parameter values, we used test problems of different sizes and structures taken from the BISON test set[29] in order to find values which give good performance across a broad range of different problem classes.

The pure ACO algorithm

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

The next parameter, β, is the one that defines the relative importance of the heuristic information as opposed to the pheromone information. From our findings, this parameter appeared to be crucial. Using the wrong value for it resulted inevitably in bad results. However, we could not find a link between any features of the problem instance and the best

value for beta. Fortunately, the good beta values for the different problems were all situated between 2 and 10, and in practice, the choice could be narrowed down to one of 2, 5 or 10. This means, though, that for every new problem these three values have to be tried out.

For the parameter k, which defines the fitness function, the results of Falkenauer[4] and Falkenauer and Delchambre[17] could be confirmed. A value of 2 was definitely better than 1. Higher values gave slightly worse results.

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

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

The hybrid ACO algorithm with local search

When the ACO is combined with local search, new parameter settings are needed. In this section, the parameters *nants*, β, ρ, γ, *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 β parameter in the algorithm with local search, it turned out that using an optimal β 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 by Dorigo and Stützle,[11] 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. There were still some very large problem instances that needed a β value of 1, but this change of value seems to be well correlated with the problem size for the hybrid ACO algorithm.

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 γ and ρ were defined. For γ, 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 ρ, the test results were very unclear. For most problems, any value between 0.1 and 0.9 gave good results. A very low ρ 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. It did, however, also cause the algorithm to get stuck in local optima from time to time. In the end, we settled for a ρ of 0.75. This gave rise to longer runs (around 30 iterations 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 needed. 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 τ_{min} is set to 0: the lower limit on pheromone values is abandoned. Also for different values $\tau(0)$, the results hardly differed in number of bins or cycles needed. Therefore, we decided to give up on the exploratory starts and set $\tau(0)$ to 0. This meant that the initial solutions were in fact created by a random form of FFD: the first item in each bin is chosen randomly with bias towards larger items (as controlled by β) and the remaining items in the bin are chosen from those left using standard FFD.

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*, and we found that a value of 4 was acceptable for all problems under consideration.

Comparing the pure ACO to other approaches

We compared our pure ACO approach for the CSP to Liang *et al*'s Evolutionary Programming approach (EP), and for the BPP to Martello and Toth's Reduction Algorithm and Falkenauer's HGGA. All tests were run on Sun Microsystems Blade 100 machines with 502 MHz UltraSPARC-IIe processors. The algorithm was implemented in Java and run under Java version 1.3.

Tests for the CSP. Liang *et al*[8] include in their paper their 20 test problems. We use their five largest single stock length problems (problems 6a–10a) to compare our approach to theirs. They have a version of their

program with and without contiguity. A CSP with contiguity is one where, apart from minimizing the number of stocks, you also want as few outstanding orders as possible. Concretely, this means that once you have started cutting items of a certain length, you want to finish all the items of that length as soon as possible. Liang *et al*'s EP with contiguity gives the best results in number of stocks, so that will be the one we compare to.

Like Liang *et al*, we did 50 independent test runs for each problem. The results are summarized in Table 1. 'FFD' gives the results obtained using the greedy FFD heuristic, 'ACO' gives the results obtained with our pure ACO approach, 'EP' gives Liang *et al*'s results. 'avg' indicates how many stocks were used on average, 'best' indicates the number of stocks in the best solution found, and 'time' indicates the average running time in CPU seconds.

Liang *et al* use a population size of 75 and a fixed number of generations for each problem. In order to compare the two approaches, we considered letting the ant algorithm build the same number of solutions as EP. However, this would have been an unfair comparison: ACO is much slower than EP at creating a single new solution, since ACO has to make the whole solution from scratch, rather than applying a fast mutation procedure to an already existing solution. Therefore, in order to get some comparison between the two approaches, we let our ant algorithm run for the same amount of time as EP on each of the 10 problems. As mentioned before, the parameter β is really crucial in our algorithm. Therefore, we had to do a few preliminary test runs for every problem to choose a good β value, which is clearly a problem for our approach.

It is clear that the ACO algorithm is at least comparable to EP from these results: it finds better solutions for four of the five problems and has the same behaviour as EP on the remaining problem 7a. The lower CPU time on problem 6a is explained by the fact that the ACO procedure terminates when it finds a solution with the known optimum number of stocks (which for problem 6a is 79).

Tests for the BPP. Falkenauer[4] compares his HGGA to Martello and Toth's Reduction Algorithm (MT). He uses 80 different uniformly generated test problems with a bin capacity of 150 and item sizes uniformly distributed between 20 and 100. He uses four different problem sizes:

Table 1 Pure ACO results for cutting stock problems

Prob	*FFD*	*EP*			*ACO*		
	Bins	*Avg*	*Best*	*Time*	*Avg*	*Best*	*Time*
6a	85	80.8	80	347	79.0	79	166
7a	71	69.0	68	351	69.0	68	351
8a	150	148.1	147	713	146.0	145	714
9a	153	152.4	152	1679	151.0	151	1652
10a	221	220.3	219	4921	218.9	218	4925

Table 2 Pure ACO results for uniform bin packing problems

Prob	*FFD*	*HGGA*		*MTP*		*ACO*	
	Bins	*Bins*	*Time*	*Bins*	*Time*	*Bins*	*Time*
u120	+14	+2	381	+2	370	+2	376
u250	+31	+3	1337	+12	1516	+12	1414
u500	+55	0	1015	+44	1535	+42	1487
u1000	+97	0	7059	+78	9393	+70	9272

120 items, 250, 500 and 1000. For each size, 20 different problems were created randomly.

Following Falkenauer, we ran our ACO algorithm once on each instance of every problem set. To get some comparison between our results and Falkenauer's, we allowed the pure ACO procedure to run for a fixed number of iterations for each set of 20 problems, such that the average time used was of the same order as the average time used by HGGA and the Martello and Toth procedure (as reported by Falkenauer).

We performed all the test runs for β values of 2, 5 and 10. For the smaller problems (120 items), the value for β did not really matter, but for the larger ones, it made a big difference. In the table below, we only report the results for the best β value. The results are summarized per problem set in Table 2. 'u120' to 'u1000' are the uniform problem sets, 'bins' indicates how far in total the solutions were above the theoretical optimum (ie the L_1 lower bound of $\lceil \Sigma i/C \rceil$ where C is the bin capacity) across the 20 problems in each set. 'time' gives the average running time in CPU seconds.

From these results, it is clear that the pure ACO algorithm is comparable to MTP but cannot beat Falkenauer's HGGA. For the small problems (size 120) it does equally well, but for the largest (size 500 and 1000), it is always sub-optimal, with an average absolute deviation from optimality of 2.1 bins for the u500 problems and 3.5 bins for the u1000 problems. This demonstrates the power of the local search procedure in HGGA and motivated us to add a similar local search to our ACO procedure.

Comparing the hybrid ACO to other approaches

We compared our hybrid ACO approach with the local search procedure added with EP, MTP and HGGA using the same problems as were used in the previous section.

Tests for the CSP. We ran the hybrid ACO procedure on the same five cutting stock problems as before, as shown in Table 3. The hybrid ACO results are compared with the results for the pure ACO approach given earlier. As before, we did 50 independent runs for each problem. For the hybrid algorithm we found that good results could be obtained within 20 000 evaluations, so we used this as the upper bound on the number of solutions the ACO procedure was allowed to build. For the tests of the pure

Table 3 Hybrid ACO results for cutting stock problems

Prob	*Pure ACO*			*HACO*			
	Avg	*Best*	*Time*	*Avg*	*Best*	*Time*	*Dev*
6a	79.0	79	166	79.0	79	0.6	0.09
7a	69.0	68	351	68.0	68	0.2	0.05
8a	146.0	145	714	143.0	143	5.2	1.8
9a	151.0	151	1652	149.0	149	10.0	4.6
10a	218.9	218	4925	215.0	215	248.9	146.4

Table 4 Hybrid ACO results for uniform bin packing problems

Prob	*HGGA*		*MTP*		*HACO*	
	Bins	*Time*	*Bins*	*Time*	*Bins*	*Time*
u120	+2	381	+2	370	0	1
u250	+3	1337	+12	1516	+2	52
u500	0	1015	+44	1535	0	50
u1000	0	7059	+78	9393	0	147
u2000	—	—	—	—	0	531
u4000	—	—	—	—	0	7190
u8000	—	—	—	—	0	43 746

ACO algorithm, typically 100 000 evaluations or more were used. The β parameter was set to 2 for all five problems.

Hybrid ACO clearly has very strong performance on these five problems: all five are reliably solved to the proven optima for these problems well within the 20 000 evaluations allowed. The standard deviation of the time taken to solve the problem over the 50 runs is also shown, to indicate the stability of the procedure across individual runs.

Tests for the BPP. We ran the hybrid ACO procedure on the same 80 bin packing problems as before, as shown in Table 4. We also generated 60 larger instances to give the ACO procedure a further test: these are generated in the same way as Falkenauer's uniform instances, but contain more items. We generated problems containing 2000, 4000 and 8000 items, with 20 random instances of each. As before, we did a single run of each problem. As with the CSP tests, we used 20 000 as the upper bound on the number of solutions the ACO procedure was allowed to build. The β parameter was set to 2 for the u120, u250, u500 and u1000 problem sets; for all the larger problem sets, a β value of 1 was found to give better performance.

The hybrid ACO procedure also does well on these problems: 138 of the 140 problems are solved to the L_1 lower bound, including three of the problems that were not solved by HGGA and all of the new larger instances. Of the remaining two problems, one (u250_13) is actually insoluble at the lower bound[30] and the remaining instance (u250_12) can be solved by the hybrid ACO procedure given a larger number of evaluations and about 2 h of CPU time.

In looking at individual results, we noticed that the instances which gave the hybrid ACO procedure the most difficulty were those which had the least spare capacity in the optimal solution, such as u250_12 (spare capacity = 11), u500_07 (spare capacity = 3), u2000_12 (spare capacity = 13) and u4000_06 (spare capacity = 5). It may well be that for these problems a less-aggressive strategy is required for both the ACO and for the local search procedure.

Memoryless experiments

Adding the local search procedure to the ACO gives a clear increase in performance, with many previously hard problems now being solved in seconds. It is therefore reasonable to ask what contribution the ACO procedure is actually making to solve the problems, and whether similar results might be obtained by applying the local search procedure repeatedly to random points in the search space until a good solution is found.

We therefore repeated the experiments from the previous section, but this time with the pheromone matrix update procedure disabled. This meant that the initial matrix entries were left undisturbed throughout the run. The results of these runs are shown in Table 5 for the cutting stock problems and Table 6 for the bin packing problems (up to size 4000).

It is clear from these results that the local search procedure can solve small problems but needs the ACO procedure when larger problems are encountered. The memoryless procedure fails to find the optimum for the

Table 5 Memoryless results for cutting stock problems

Prob	*HACO*			*No memory*		
	Avg	*Best*	*Time*	*Avg*	*Best*	*Time*
6a	79.0	79	1	79.0	79	24
7a	68.0	68	1	68.0	68	1
8a	143.0	143	5	144.0	144	1064
9a	149.0	149	10	150.0	150	997
10a	215.0	215	249	216.8	216	1707

Table 6 Memoryless results for uniform bin packing problems

Prob	*HACO*		*No memory*	
	Bins	*Time*	*Bins*	*Time*
u120	0	1	0	1
u250	+2	52	+6	166
u500	0	50	+5	432
u1000	0	147	+10	1850
u2000	0	531	+43	19 286
u4000	0	7190	+118	131 137

three largest cutting stock problems and for many of the larger bin packing problems (including all of the u2000 and u4000 instances).

The results shown here are from biased initial points: the initial $\tau(i, j)$ entries were set to 0 and β was set to the same values as before. This generates solutions in which the first item in each bin is chosen with a probability proportional to $\eta(j)^{\beta}$ and all subsequent items are chosen to be the largest which will still fit in the bin. This results in FFD-like solutions, albeit with some random variation given by the probabilistic choice of the first item for each bin. We also tried setting the initial $\tau(i, j)$ entries to 1 and β to zero, to give the local search entirely random starting points; however, the results generated were worse than those shown in Tables 5 and 6.

Conclusions and further work

This paper has presented an ACO approach for the BPP and the CSP. 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. The relative importance of the pheromone trail information as opposed to the heuristic information is defined by the parameter β, and is crucial for good performance of the pure ACO algorithm.

We also present 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 managed to solve all but one of the benchmark problems under consideration in reasonable time, and was capable of solving the last given an extended run.

When compared to existing evolutionary approaches, the pure ACO algorithm was comparable with Liang *et al*'s EP solution for the CSP and Martello and Toth's MTP for the BPP. The pure ACO algorithm failed to compete with Falkenauer's HGGA. The hybridized ACO algorithm was much faster and could outperform EP, MTP and HGGA on the problems under consideration.

We are currently testing the hybrid ACO algorithm on a wider variety of test problems, including Falkenauer's triplet problems and the full BISON test set.[29] The initial indications are that the hybrid ACO procedure has variable performance, depending on the class of benchmark problem being tackled. For the majority of problems, it can be competitive with or even outperform the best-known solution procedures for the CSP and BPP, such as Alvim *et al*'s Hybrid Improvement Heuristic[31] and Fleszar and Hindi's Perturbation MBS′ + VNS.[32] However, it fails to perform as well as other procedures at artificially constructed zero-slack problems, such as Falkenauer's triplets. We plan to write a future article giving these results in full and analysing the way in which the hybrid ACO algorithm navigates the solution space.

The ACO approach to the bin packing and cutting stock problems presented here is quite generic, and is based on the idea of reinforcement of good groups through binary couplings of item sizes. The approach used should be capable of being adapted to solve similar grouping problems, such as the restricted BPP (where some pairs of items cannot be placed in the same bin), the CSP with multiple stock sizes, the multiprocessor scheduling problem, and simple timetabling problems. We are currently adapting the algorithm presented here to solve some of these problems.

Acknowledgements—We would like to thank Ko-Hsin Liang and Xin Yao for sharing their results with us. We would also like to thank Thomas Stützle, Gianni di Caro and Luca Gambardella for useful feedback on this work.

References

1 Dyckhoff H (1990). A typology of cutting and packing problems. *Eur J Opl Res* **44**: 145–159.
2 Martello S and Toth P (1990). *Knapsack Problems, Algorithms and Computer Implementations*. John Wiley and Sons Ltd: England.
3 Haessler RW and Sweeney PE (1991). Cutting stock problems and solution procedures. *Eur Journal Opl Res* **54**: 141–150.
4 Falkenauer E (1996). A hybrid grouping genetic algorithm for bin packing. *J Heuristics* **2**: 5–30.
5 Hinterding R and Khan L (1995). Genetic algorithms for cutting stock problems: with and without contiguity. In: Yao X (ed). *Progress in Evolutionary Computation*. Springer: Berlin, Germany, pp 166–186.
6 Reeves C (1996). Hybrid genetic algorithms for bin-packing and related problems. *Ann Oper Res* **63**: 371–396.
7 Vink M (1997). Solving combinatorial problems using evolutionary algorithms. Available from http://citeseer.nj.nec.com/vink97solving html.
8 Liang K-H, Yao X, Newton C and Hoffman D (2002). A new evolutionary approach to cutting stock problems with and without contiguity. *Comput Oper Res* **29**: 1641–1659.
9 Dorigo M, Maniezzo V and Colorni A (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE T Syst Man Cy B* **26**: 29–41.
10 Bonabeau E, Dorigo M and Theraulez G (1999). *Swarm Intelligence: From Natural to Artificial Intelligence*. Oxford University Press, Inc.: New York, NY, USA.
11 Dorigo M and Stützle T (2002). The ant colony optimization metaheuristic: algorithms, applications, and advances. In: Glover F and Kochenberger G (eds). *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, USA, pp 251–285.
12 Bilchev G (1996). Evolutionary metaphors for the bin packing problem. In: Fogel L, Angeline P and Bäck T (eds). *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*. MIT Press, Cambridge, MA, USA, pp 333–341.
13 Bischoff EE and Wäscher G (1995). Cutting and packing. *Eur J Opl Res* **84**: 503–505.

14 Coffman EG, Garey MR and Johnson DS (1996). Approximation algorithms for bin packing: a survey. In: Hockbaum D (ed). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, Boston, MA, USA, pp 46–93.
15 Gilmore PC and Gomory RE (1961). A linear programming approach to the cutting stock problem. *Oper Res* **9**: 848–859.
16 Winston WL (1993). *Operations Research: Applications and Algorithms*. International Thompson Publishing: Belmont, CA, USA.
17 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*. IEEE Press, Piscataway, NJ, USA, pp 1186–1192.
18 Dorigo M and Gambardella L (1997). Ant colony system: a cooperative learning approach to the travelling salesman problem. *IEEE T Evolut Comput* **1**: 53–66.
19 Stützle T and Hoos H (2000). MAX-MIN ant system. *Future Gener Comp Sys* **16**: 889–914.
20 Maniezzo V, Colorni A and Dorigo M (1994). The ant system applied to the quadratic assignment problem. Technical Report, IRIDIA, Universit é Libre de Bruxelles, Brussels, Belgium.
21 Gambardella L, Taillard E and Dorigo M (1999). Ant colonies for the quadratic assignment problem. *J Opl Res Soc* **50**: 167–176.
22 Bauer A, Bullnheimer B, Hartl R 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*. IEEE Press, Piscataway, NJ, USA, pp 1445–1450.
23 Stützle T (1998). An ant approach to the flow shop problem. In: *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing*. Verlag Mainz, Aachen, Germany, pp 1560–1564.
24 Gambardella L, Taillard E and Agazzi G (1999). MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M and Glover F (eds). *New Ideas in Optimization*. McGraw-Hill, London, UK, pp 63–76.
25 Costa D and Hertz A (1997). Ants can colour graphs. *J Opl Res Soc* **48**: 295–305.
26 Michel R and Middendorf M (1999). An ACO algorithm for the shortest supersequence problem. In: Corne D, Dorigo M and Glover F (eds). *New Ideas in Optimization*. McGraw-Hill, London, UK, pp 51–61.
27 Leguizamon G and Michalewicz Z (1999). A new version of ant system for subset problems. In: *Proceedings of the 1999 Congress of Evolutionary Computation*. IEEE Press, Piscataway, NJ, USA, pp 1459–1464.
28 Alvim A, Glover F, Ribeiro C and Aloise D (1999). Local search for the bin packing problem. Available from http://citeseer.nj.nec.com/alvim99local.html.
29 Scholl A, Klein R and Jürgens C (1997). BISON: a fast procedure for exactly solving the one-dimensional bin packing problem. *Comput Oper Res* **24**: 627–645 Test problems available from http://www.bwl.tudarmstadt.de/bw13/forsch/projekte/binpp.
30 Gent I (1997). Heuristic solution of open bin packing problems. *J Heuristics* **3**: 299–304.
31 Alvim A, Glover F, Ribeiro C and Aloise D (2002). A hybrid improvement heuristic for the bin packing problem. Available from http://citeseer.nj.nec.com/557429.html.
32 Fleszar K and Hindi K (2002). New heuristics for one-dimensional bin-packing. *Comput Oper Res* **29**: 821–839.

Received September 2002;
accepted March 2004 after one revision