
Genetic Programming and Evolvable Machines, 3, 345–361, 2002

Fast Ant Colony Optimization on Runtime Reconfigurable Processor Arrays

DANIEL MERKLE AND
MARTIN MIDDENDORF
{merkle,middendorf}@informatik.uni-leipzig.de
Parallel Computing and Complex Systems Group, Faculty of Mathematics and Computer Science, University of Leipzig, Ostenstr. 26-28, D-04109, Leipzig, Germany

Submitted August 10, 2001; Revised April 1, 2002

Abstract. Ant Colony Optimization (ACO) is a metaheuristic used to solve combinatorial optimization problems. As with other metaheuristics, like evolutionary methods, ACO algorithms often show good optimization behavior but are slow when compared to classical heuristics. Hence, there is a need to find fast implementations for ACO algorithms. In order to allow a fast parallel implementation, we propose several changes to a standard form of ACO algorithms. The main new features are the non-generational approach and the use of a threshold based decision function for the ants. We show that the new algorithm has a good optimization behavior and also allows a fast implementation on reconfigurable processor arrays. This is the first implementation of the ACO approach on a reconfigurable architecture. The running time of the algorithm is quasi-linear in the problem size n and the number of ants on a reconfigurable mesh with n^2 processors, each provided with only a constant number of memory words.

Keywords: ACO, reconfigurable architectures, quadratic assignment

1. Introduction

Ant Colony Optimization (ACO) is a metaheuristic that has been applied successfully to solve various combinatorial optimization problems (for an overview see [6]). In ACO, several generations of artificial ants search for good solutions. Every ant of a generation builds up a solution step by step, thereby going through several decisions until a solution is found. Similar to real ants, the artificial ants that found a good solution mark their paths through the decision space by putting some amount of pheromone on the edges of the path. The following ants of the next generation are attracted by the pheromone so that they will search in the solution space near good solutions.

Some authors have studied parallel versions of ACO algorithms (a short overview is given in [19]). All these authors assume that a processor can hold a whole problem instance and the whole pheromone information. A standard approach is that every processor holds a colony of ants and after every generation the colonies exchange information about their solutions. Then, every colony computes the new pheromone information which is usually stored in some pheromone matrix. The parallel implementations following this approach differ mainly in granularity. One approach is to do the computations for the new pheromone matrix locally in the colonies. Whereas in another approach these computations are done centrally by a master processor

which distributes the new matrix to the colonies. Exceptions are [4, 17, 19] where the colonies exchange information only after several generations of ants.

In this paper we study the problem of implementing the ACO heuristic on large processor arrays with small processors that have only a constant number of registers. Processor arrays with a dynamically reconfigurable bus sytem are especially attractive for our purpose since they allow us to perform some basic algorithmic tasks like bit-oring, bit-summation, and finding the rank of a number in a set of numbers much faster than on most other parallel architectures (compare e.g. [18, 20, 21]).

In order to find a fast implementation we propose some changes to a standard form of ACO algorithms that is used for solving permutation problems. One major change is to give up the generational principle which is used in every ACO algorithm that has been proposed so far. The new ACO approach allows us to pipeline the ants through the processor array. Note, that this idea might be interesting in general for ant algorithms. Another major change is that an ant will no longer use the pheromone values directly to determine the probabilities of the possible outcomes of a decision (as has been done in nearly all ant algorithms proposed so far). Instead, it uses a threshold function that assigns to every possible outcome of the next decision either a high or a low probability, depending on whether the pheromone value is above or below the threshold. This enables us to use a variant of the fast bit-summation algorithm of [18] for realizing the decisions of an ant. We show that the new ACO algorithm, called s-m-p-ANT, can be implemented in quasi-linear time (with respect to the total number of ants that search for a solution and the problem size) on a reconfigurable mesh with n^2 processors. This has to be compared to a sequential running time of $O(zn^2)$ where n is the problem size and z is the total number of ants.

In Section 2, the model of a reconfigurable mesh used in this paper is explained. Section 3 introduces the basic ACO algorithm. Some operations on reconfigurable meshes that are used by our ACO algorithm are considered in Section 4. Section 5 explains the new ACO algorithm that allows a fast implementation on reconfigurable meshes. Implementation details and a run time analysis are given in Section 6. Section 7 contains a description of the test problems and a performance evaluation. A conclusion is given in Section 8.

2. Model of computation

The reconfigurable mesh (RM) is a well studied model for run-time reconfigurable architectures (see [3] for a short overview). An RM consists of a set of processing elements (PE's) arranged on a $k \times n$ grid. Every PE contains four ports (named north, east, south, and west port) enabling it to connect to its neighbors. The linked ports construct the static topology. Every PE is furthermore equipped with a number of switched lines, which link the PE's ports internally. The PE's configuration is set by these switches, which essentially form the dynamic topology through buses as depicted in Figure 1.

The PEs work synchronously. Every PE can read from and write to the bus lines it is connected to, that is, we have concurrent read, concurrent write buses (CRCW-buses). For this paper, we need each horizontal connection to have $\max\{\log(k \cdot n),$

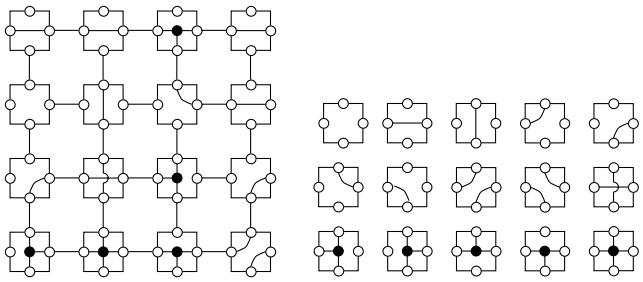

Figure 1. Reconfigurable mesh with 16 PEs (left), possible connections of ports within a PE (right).

$\log x\}$ bit-lines and each vertical connection has $\max\{\log^2(k \cdot n), \log x\}$ bit-lines where x is the largest number the algorithm has to send over the bus. When several PEs write to the same bus the result is the bitwise OR. If no PE writes on a bus then its value is 0. Every PE has only a constant number of registers and it knows its row and column indices. Within one time step every PE can locally configure the bus, write to and/or read from one of the buses it is connected to, and perform some local computation. Signal propagation on buses is assumed to take constant time regardless of the number of switches on the bus. This is the standard assumption for this model of computation (e.g. [20]).

3. Ant colony optimization

In this section we describe a generic ACO algorithm for permutation problems where the ants use a place-item pheromone matrix to find the permutation. Such an approach has been used for the Quadratic Assignment problem (QAP) (see [23] for an overview of ACO for the QAP) and for several permutation scheduling problems like the Flow-shop problem [22], the Resource-Constraint Project Scheduling problem (RCPSP) [15], and the Single Machine Total (Weighted) Tardiness problem (SMTWTP) [1, 2, 14]. The successfulness of ACO for the QAP was shown by several authors ([7, 9, 11, 13, 12, 24]). Also for various permutation scheduling problems good results have been obtained with ACO. In [15] it was shown that the ACO approach performed better on the average over the 600 largest benchmark RCPSP-instances from [26] than 12 other heuristics (including simulated annealing, tabu sarch, and genetic algorithms) that have been tested in [10]. In [2] a comparison between ACO and other heuristics on a set of benchmark problems from [25] for the SMTWTP was done. ACO was able to find, for all 125 test instances with 100 jobs, the best known solutions. This was significantly better than the best known Tabu Search method and other heuristics for SMTWTP. Only iterated dynasearch reached a similar performance as ACO [5].

We explain how our ACO algorithm works on the basis of the Quadratic Assignment problem. For general permutation problems one has to find a permutation of a set of n given items so that the permutation has a minimal value with respect to a given evaluation function. In the case of the QAP the problem is to find for a set of n facilities, a set of n locations, distance matrix $D = [d_{ij}]$, and flow matrix

$F = [f_{hk}]$ an assignment π of facilities to locations so that the sum of all products of flows between facilities by the distance between locations is minimized:

$$\min\left(\sum_{i,j}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)}\right)$$

The ACO algorithm works as follows. In every generation each of $m \leq n$ ants constructs one solution. Every ant selects the items in the order in which they will appear in the permutation. For the selection of an item an ant uses heuristic information as well as pheromone information. The heuristic information, denoted by η_{ij}, and the pheromone information, denoted by τ_{ij}, are indicators of how good it seems to have item j at place i of the permutation. The heuristic value is generated by some problem dependent heuristic whereas the pheromone information stems from former ants that have found good solutions.

The next item is always chosen by an ant according to the following rule which has been called the Pseudo-Random-Proportional Action Choice Rule ([8]). With probability q_0, where $0 \leq q_0 \leq 1$ is a parameter of the algorithm, the ant chooses an item j from the set $\mathcal{S}$ of items that have not been selected so far which maximizes

$$\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} \tag{1}$$

where α and β are constants that determine the relative influence of the pheromone values and the heuristic values on the decision of the ant. With probability $1 - q_0$ the next item is chosen from the set $\mathcal{S}$ according to the probability distribution that is determined by

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in \mathcal{S}} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}} \tag{2}$$

Often, in case of the QAP no heuristic values η_{ij} are used. The ants that found the m' for a fixed $m' \leq m$ best solutions in a generation are then allowed to update the pheromone values. In addition, a so called elitist ant is allowed to update the pheromone values according to the best solution found so far.

But before the pheromone update is done some of the old pheromone is evaporated according to

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} \tag{3}$$

The reason for this is that old pheromone should not have a too strong influence on the future. Then, every ant that is allowed to update does the following: for every item j in the permutation of its solution some amount Δ of pheromone is added to element τ_{ij} of the pheromone matrix where i is the place of item j in the permutation (i.e. $\tau_{ij} = \tau_{ij} + \Delta$). The algorithm stops when some stopping criterion is met, for example, a certain number of generations has been done.

It has to be mentioned that we do describe all variants of ACO algorithms that have been proposed for the QAP problem to concentrate on the general behavior of ACO algorithms. For example, we do not use local pheromone update, that is, the ants do not change the pheromone values during their search for a solution. Local update is an option that is used in some ACO algorithms for the QAP (e.g., [24]). It should be mentioned that our result about the run time can also be proved for using local pheromone update. Moreover, we do not use local optimization on the solutions that have been found by the ants.

4. Basic operations on the RM

The following result is well known for RMs.

Lemma 1. ([20]) *Let integers a_i, $i \in [1:n]$ be stored in a processor in row i of an $n \times n$ RM. The maximum of $\{a_i \mid i \in [1:n]\}$ can be determined in time $O(1)$.*

We give a sketch of the algorithm since it illustrates standard methods for RMs. The first step is to send for all $i, j \in [1:n]$ the integers a_i and a_j to PE P_{ij} in time $O(1)$ using column and row buses. Then every PE sets a flag if $a_j > a_i$ and otherwise the flag is not set. Element a_j is the maximal element when every flag in column j is one. This can be checked in time $O(1)$ as follows: Every PE with the flag set connects its north and south port, then every PE P_{nk} with $k \in [1:n]$ sends a signal on its south port and every PE P_{1k} with $k \in [1:n]$ reads from its north port. When PE P_{1j} reads a signal on its north port then a_j is the maximum value.

The following theorem has been shown in [18]. Note, that a similar result for an $n \times n$ RM has been shown independently in [21] using similar techniques.

Theorem 1. ([18]) *The sum of $k \cdot n$ bits can be computed in time $O(\log^* k + \log n/\sqrt{k \log k})^1$ time on a reconfigurable mesh of size $k \times n$.*

In this paper we need a slightly stronger result. In particular we need the sum of the bits in every submesh containing the first $i \cdot k$ columns for $i \in [1:l]$, where without loss of generality $n = l \cdot k$ for some integer l. Using the same techniques as in the proof of Theorem 1 we can prove the following theorem.

Theorem 2. *Given a $k \times n$ RM with $r = n/k$ where every processor stores one bit. The sum of $i \cdot k^2$, $i \in [1:r]$ bits in all submeshes containing the first $i \cdot k$ columns can be computed in time $O(\log^* k + \log n/\sqrt{k \log k})$.*

5. ACO on the RM

The general principle of our ACO algorithm on the RM is to embed the $n \times n$ pheromone matrix M into an $n \times n$ RM so that PE P_{ij} contains only the pheromone value τ_{ij}, $i, j \in [1:n]$. The ants are then pipelined through the RM.

A possible way to proceed could be as follows. The first ant starts in row 1 of the RM and selects the first item. Then it moves to row 2 and selects the second item. When an ant moves it takes the information with it which items have already been selected, that is, items which are not in the set of selectable items $\mathcal{S}$. This process continues until the ant reaches row n where it has determined its permutation. The next ant always follows its predecessor ant one row behind.

In more detail, the selection of an item in a row i by an ant is done as follows. Every PE $P_{i-1,j}$ in row $i-1$ knows whether item j has been selected in one of the rows $1, \ldots, i-1$. PE $P_{i-1,j}$ sends this information to P_{ij} when the ant moves to row i. The prefix-sums of the $\tau_{ij} \cdot \eta_{ij}$ values from all PEs in $\mathcal{S}$ are determined. Formally, for $\mathcal{S} = \{j_1, j_2, \ldots, j_{n-(i-1)}\}$, $j_1 < j_2 < \cdots < j_{n-(i-1)}$ the prefix-sums of the $\tau_{ij} \cdot \eta_{ij}$ values are all sums $\tau_{ij_1} \cdot \eta_{ij_1} + \tau_{ij_2} \cdot \eta_{ij_2} + \cdots + \tau_{ij_k} \cdot \eta_{ij_k}$ with $k \in [1 : n-(i-1)]$. The first PE in the row chooses a random number z from the interval $[0, \sum_{j \in \mathcal{S}} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta})$. Random number z is then sent to all PEs in the row. PE P_{ij} is selected when z is in the interval $[\sum_{l<j,\, l \in \mathcal{S}} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}, \sum_{l \leq j,\, l \in \mathcal{S}} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta})$. Clearly, every PE P_{ij} can determine in time $O(1)$ whether it is selected or not by using its own prefix-sum and the prefix-sum of the next PE to the left with a column index in $\mathcal{S}$.

When all ants in a generation have found their solution it is determined which ants are allowed to update the pheromone information. Then pheromone evaporation and update are done and the next generation of ants starts. To force the ants to search more near the best solution that has been found so far it is possible to introduce an elitist ant. The elitist ant is allowed to update the pheromone information according to the best solution found so far in every generation.

Since the prefix-sums can be determined in time $O(\log n)$ in every row it, is not hard to see that the algorithm will run on an $n \times n$ RM in time $O(x \cdot (m+n) \cdot \log n)$ where x is the number of generations.

In the following, we propose some changes to this ACO algorithm that will allow for a faster implementation on the RM but does not decrease the optimization behavior in a significant way or decreases it at most slightly (when the same number of ants is used) as we show by experiments. The proposed changes are also interesting by themselves, and not only when a parallel implementation is needed. Let the standard generational ACO algorithm be called generational-ANT (g-ANT). Pipelining-ANT (p-ANT) is the new ACO algorithm that uses a non-generational approach (as described in the following Subsection 5.1) and allows to better pipeline the ants through the mesh. The ACO algorithm that uses the non-generational approach and the modified Pseudo-Random-Proportional Action Choice Rule as described in Subsection 5.2 is called max-p-ANT (m-p-ANT). The version of m-p-ANT that also uses the simplified probability distribution for choosing the next item as described in Subsection 5.3 is called simple-m-p-ANT (s-m-p-ANT).

5.1. p-ANT: Non-generational pheromone update

A generation of ants is a number of ants that usually construct their solutions using the same pheromone information and from which the best individuals for pheromone update are selected. For the ACO algorithm for the RM we abandon

the principle of using generations of ants. Then we can produce a constant flow of ants by pipelining one ant after the other through the mesh. Without the use of generations we need a new selection criterion to decide which ants are allowed to update the pheromone information. The criterion that is applied here after an ant has found a solution is whether the solution found by the ant is better than the $m' - 1$ solutions of the preceding $m - 1$ ants, $m' \leq m$. A similar approach for deciding whether an ant is allowed to update within a generational ACO algorithm was proposed by Maniezzo [11]. He used positive and negative pheromone update depending on whether the solution of an ant was better or worse than the average solution of a fixed number of preceding ants. In [16] an update scheme was used were every ant is allowed to update and evaporate a certain amount of pheromone directly after is has found a solution.

Observe that our new pheromone update criterion does not allow for determining the number of ants that update in advance. Further observe, that the pheromone update is done by an ant while other ants are pipelined through the mesh constructing a solution. This principle—immediate update of the pheromone matrix when new information is available—might also be useful for other parallel/distributed implementations of ACO algorithms. Evaporation is done every time an ant is allowed to update the pheromone information.

One problem with this pheromone update criterion is that the preceding ants have worked on older pheromone values and therefore might have worse chances of finding good solutions. This is in contrast to the case when solutions of ants are compared that are in the same generation. Hence the new update criterion is likely to be milder than in the original ACO algorithm. Therefore we modify the update process as follows. Every ant waits until the following $\lfloor (m-1)/2 \rfloor$ have found a solution. Then the ant is allowed to update when its solution is better than the $m' - 1$th best solutions that have been found by the preceding $\lceil (m-1)/2 \rceil$ ants and the following $\lfloor (m-1)/2 \rfloor$ ants. In a series of ants with equal solutions we allow only every (m/m')th ant to update. Observe, that with this modification the distance between two updating ants is at least $\lfloor (m-1)/2 \rfloor + 1$ compared to minimal distance 1 when using the unmodified version (the average distance is m for the generational algorithm).

Pipelining the ants using this non-generational approach allows one to obtain a running time of $O((z+n) \cdot \log n)$ instead of $O(x \cdot (m+n) \cdot \log n)$ for the generational approach when $z = x \cdot m$ is the total number of ants. The changes proposed in the following two subsections are done to reduce the $\log n$-factor in the running time.

5.2. *m-p-ANT: Modified pseudo-random-proportional action choice rule*

In the ACO algorithm described in Section 3 an ant chooses the next item via exploitation with probability q_0, that is, according to Formula (1). In order to obtain a fast implementation we use a modified method here. When the number of selectable items is larger than a constant $r \leq 1$ the ant randomly selects r items from the set of available items. From these items it chooses that one with the maximal value $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$.

5.3. *s-m-p-ANT: Simplified probability distribution*

Another feature of our ACO algorithm is that the ants do not directly use the $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$ values when determining the next item according to Formula (2). It was shown in [24] that it can be advantageous to restrict the range of possible pheromone values to lay between some minimal and some maximal value. Then every selectable item has at least some small probability of being chosen by an ant. Here we go one step further and use only two (or a constant number of) different probabilities for the selectable items in every decision of an ant. Then it is enough to have one bit for each item per row which determines whether it has a low or high probability to be chosen. The advantage of this method is that it is enough to compute the prefix-sum of these bits instead of the prefix-sum of real numbers to determine which item is selected. Whether an item receives the high or the low probability is determined by a threshold function. Formally, let threshold $t > 0$, and parameters $h > l > 0$. For each item $j \in \mathcal{S}$ define

$$g(\tau_{ij} \cdot \eta_{ij}) := \begin{cases} h & \text{if } \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} > t \\ l & \text{otherwise} \end{cases} \tag{4}$$

In PE P_{ij}, for each $j \in \mathcal{S}$ a bit l_{ij} is set to one if $g(\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}) = l$. Otherwise, a bit h_{ij} is set to one. Now, the number n_l of PEs in the row that have the l-bit set is determined. Also the number n_h of PEs that have the h-bit set is determined. The random number z which determines the next item is selected from the interval $[0, n_l \cdot l + n_h \cdot h)$. If $z \in [(p-1) \cdot l, p \cdot l)$ the pth PE in the row with an l-bit that has been set to one is selected. If $z \in [n_l \cdot l + (p-1) \cdot h, n_l \cdot l + p \cdot h)$ then the pth PE in the row with an h-bit that has been set to one is selected. The values of t, h, and l might change during the run of the algorithm (details are described later).

For the Modified Pseudo-Random-Proportional Action Choice Rule an ant that has to select r items from the set of available items prefers items with $f(\tau_{ij} \cdot \eta_{ij}) = h$. From the at most r (selected) items the ant chooses the item with maximal value of $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$.

6. Implementation and running time

In this section we describe some implementation details of algorithms s-m-p-ANT, m-p-ANT, and p-ANT on the RM. The aim is to find a fast implementation that works for large meshes.

6.1. *Choosing the next item*

In order to be able to compute the prefix-sums of the l- and h-bits for the selection of an item according to the methods described in Subsections 5.2 and 5.3 we use an $n \log^2 n \times (n/\log^2 n)$ RM (It is assumed in this section that that $n > \log^2 n$). The n PEs that contain the pheromone values of one row of the pheromone matrix form

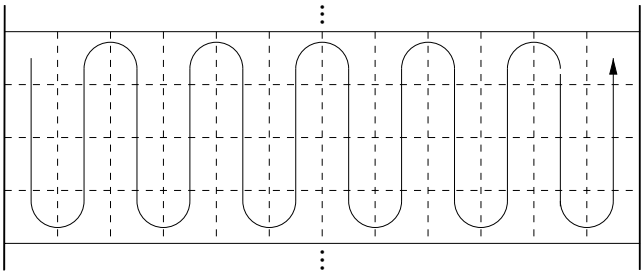

Figure 2. Embedding of a row of the pheromone matrix in a submesh of the RM.

a $\log^2 n \times (n/\log^2 n)$ submesh (see a sketch of a submesh with the embedding of a row in Figure 2).

For choosing the next item according to the simplified pheromone distribution described in Subsection 5.3 we first determine the total number n_l and n_h of l- respectively h-bits that are set to one in the corresponding $n\log^2 n \times (n/\log^2 n)$ submesh. According to Theorem 1 this can be done in time $O(\log^* n)$. Then a random number is chosen according to the method described in Section 5.2. Assume without loss of generality, that we need to find the pth PE with an l-bit that is set to one. By Theorem 2 it follows that the sums of all l-bits in the first $i \cdot \log^2 n$ columns, $i \in [1 : (n/log^4 n)]$ of a submesh can be determined in time $O(\log^* n)$. It is now clear in which part of size $\log^2 n \times \log^2 n$ of the submesh the sought PE is. It remains to find the p'th PE with an l-bit that is set to one in this submesh. With a similar technique we again determine in time $O(\log^* n)$ in which subpart of size $\log n \times \log n$ the sought PE is and it remains to find the p''th PE with an l-bit that is set to one in this subpart. This can be done in time $O(1)$. Altogether, it follows that the decision of an ant according to the simplified probability distribution can be done in time $O(\log^* n)$. Using essentially the same techniques, it is also possible to choose the next item according to the Modified Pseudo-Random-Proportional Action Choice Rule as described in Subsection 5.2 in time $O(\log^* n)$.

6.2. Computing solution quality

In this paper we assume that the quality of a solution can be computed by an ant during solution construction in time $O(1)$ per decision (time $O(\log^* n)$ per decision works also). For the QAP this can be done when the instance is sparse, that is, either the flow matrix $F = [f_{hk}]$ or the distance matrix $D = [d_{ij}]$ has at most a constant number of nonzero elements in every row. Note that sparse matrices occur often in real world QAP instances. Without loss of generality we assume that the distance matrix $D = [d_{ij}]$ has at most a constant number of nonzero elements in every row. Then flow matrix F is directly mapped onto the RM, that is, the processor that stores pheromone element τ_{ij} stores also f_{ij}. Since $D = [d_{ij}]$ is sparse it has only $O(n)$ nonzero elements. This allows us to store D every constant number of neighbored $\log^2 n \times (n/\log^2 n)$ submeshes in the RM. Now, it is not hard to show with standard operations on the RM that after every decision of the ant it takes

time $O(1)$ to update the partial sum of product flows between the already assigned items by the distance with their corresponding locations.

6.3. Pheromone update

In order to be able to perform the pheromone update of an ant in time $O(1)$ a decision of an ant (i.e., the selection of a job) is stored in a PE in the corresponding row. At every time step the decisions of at most $n + m/2 \leq \frac{3}{2}n$ ants needs to be stored in the RM. Therefore we can assign to each processor of the mesh at most two ants for storing their decisions.

To decide whether an ant is allowed to update the pheromone information the qualities of the solutions of the $\lceil (m-1)/2 \rceil$ preceding ants and of the $\lfloor (m-1)/2 \rfloor$ following ants have to be known. Hence, the solution qualities of the $(3/2)(m-1)+1 \leq n$ recent ants that finished computing a solution have to be stored in the mesh. Together with the solution qualities their ranks are also stored. These ranks have to be updated when an old solution is omitted and a new solution has to be considered. This can be easily done in time $O(1)$ as follows. For each solution that is stored: i) when the solution that is omitted was better than our solution the rank of our solution improves by 1, otherwise it does not change; ii) when the new solution is better than our solution the rank of our solution decreases by 1, otherwise it does not change. The rank of the newly added solution can be computed in time $O(\log^* n)$: the quality of the new solution is compared with all other solution qualities in parallel, and a bit is set when it is worse. Addition of the resulting bits gives the rank. If the rank of the new solution is high enough, the corresponding ant is allowed to update.

An ant that is allowed to update sends a signal to all PEs. Pheromone evaporation is done in every PE by decreasing its pheromone value according to Formula (3). Then the PEs that store a decision of the ant send the decision to every PE in their row. If the ith decision was to map item (location) j to place (facility) i then the PE that stores τ_{ij} adds the amount Δ of pheromone to its pheromone value.

6.4. Running time

We obtain the following theorem:

Theorem 3. *Algorithm s-m-p-ANT for the QAP with n items, n locations, and where either the flow matrix or the distance matrix is sparse runs on an $n \log^2 n \times n/\log^2 n$ RM in time $O((z+n) \cdot \log^* n)$ which is quasi-linear in $\max\{z, n\}$ where z is the total number of ants.*

This running time of s-m-p-ANT has to be compared to a sequential running time of the standard ACO algorithm of $O(z \cdot n^2)$ where n is the problem size and $z = x \cdot m$ is the total number of ants. Since s-m-p-ANT is a generic algorithm the same result also holds for other problems when the implementation can be done

similarly as for QAP (e.g., the solution quality can be computed using only time $O(1)$ per decision). When the QAP instances are not sparse, additional running time has to be spent to compute the solution quality.

7. Parameters and test results

We tested the optimization behavior of algorithms s-m-p-ANT, m-p-ANT, p-ANT, and g-ANT on QAP instances tai80b, tai64c, and sko81 from the benchmark library QAP-LIB. These are instances from the QAP-LIB that are reasonably sparse and not too small. Run times are not given in this section because the tests were done with a sequential implementation. The parameters used for the test runs are: $\alpha = 1$, $\beta = 0$, $\rho \in \{0.95, 0.98, 0.99, 0.995\}$, $q_0 \in \{0, 0.1, 0.2, \ldots, 1.0\}$. The number of ants in every generation of the standard algorithm was $m = 10$. The total number of ants in a run was 250000 (i.e. 25000 generations in case of the generational ant algorithm g-ANT). A run was stopped when the average solution quality of the last m ants had not changed for 500 times (respectively for 50 generations in case of g-ANT). Each result is averaged over 10 runs. Every test run was done twice—once without an elitist ant and once with an elitist ant. The elitist ant has the same influence as the updating ant, that is, it is allowed to add the same amount of pheromone. Since the results with and without the elitist ant were quite similar, we describe in the following, only the results obtained without using the elitist ant.

Every row i of the pheromone matrix has its own threshold values t and h. Threshold t for row i is determined as half of the average pheromone value in the row, that is, $t = (\sum_{j=1}^{n} \tau_{ij})/2n$. In every row we set $l = 1$. The value h is changed dynamically. Let $\mathcal{H}$ be the set of locations for which the corresponding pheromone value is higher than the average pheromone value in the row, that is, locations l with $\tau_{il}^{\alpha} > (\sum_j \tau_{ij}^{\alpha})/n$, and let $\mathcal{L}$ be the other locations in a row. Then for row i the value of h is defined as follows

$$h = \frac{\sum_{l \in \mathcal{H}} \tau_{il}^{\alpha}/|\mathcal{H}|}{\sum_{l \in \mathcal{L}} \tau_{il}^{\alpha}/|\mathcal{L}|}$$

Note, that it is possible to determine the values t and h during the run of the algorithm in time $O(1)$ before every decision of an ant.

A heuristic was used to determine the order in which the facilities were assigned to the locations. According to this heuristic the facilities are ordered by their flow values (i.e. $\sum_j f_{ij}$).

The test results for the three QAP instances tai80b, tai64c, and sko81 for different values of q_0 and $\rho = 0.98$ are shown in Figures 3–5. Every figure contains a curve for s-m-p-ANT, m-p-ANT, p-ANT, and g-ANT. It can be seen that algorithm g-ANT performs quite similar to the non-generational algorithm p-ANT. For both of these algorithms the solution quality becomes worse the higher the value is of q_0. The solution qualities found by algorithm m-p-ANT, which uses the Modified Pseudo-Random-Proportional Action Choice Rule, do not depend much on the value of q_0. Note, that m-p-ANT equals p-ANT for $q_0 = 0$ and m-p-ANT equals s-m-p-ANT for

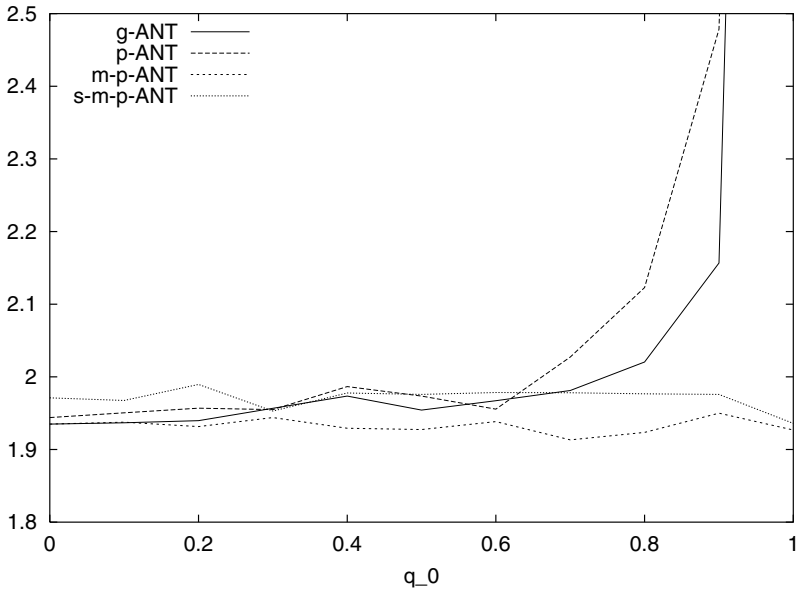

Figure 3. Solution quality (divided by 10^6) of algorithms g-ANT, p-ANT, m-p-ANT, and s-m-p-ANT for QAP instance tai64c for different values of q_0.

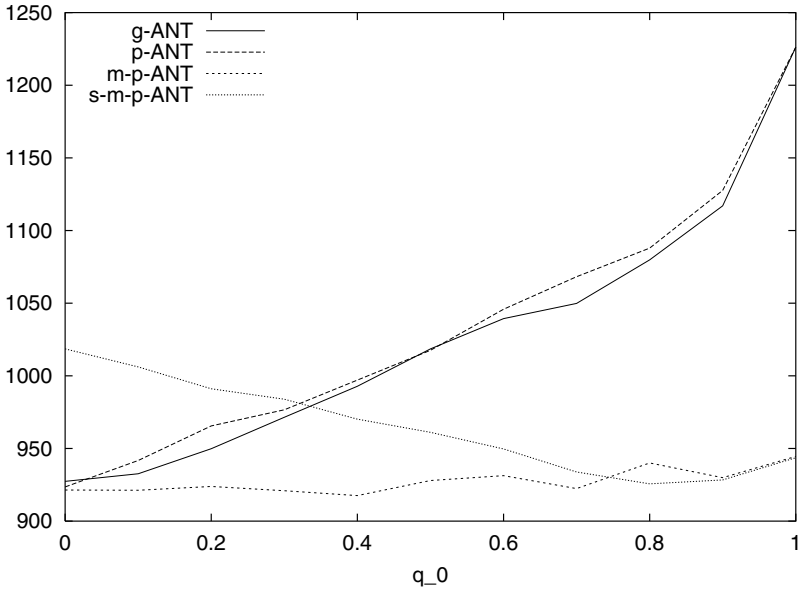

Figure 4. Solution quality (divided by 10^6) of algorithms g-ANT, p-ANT, m-p-ANT, and s-m-p-ANT for QAP instance tai80b for different values of q_0.

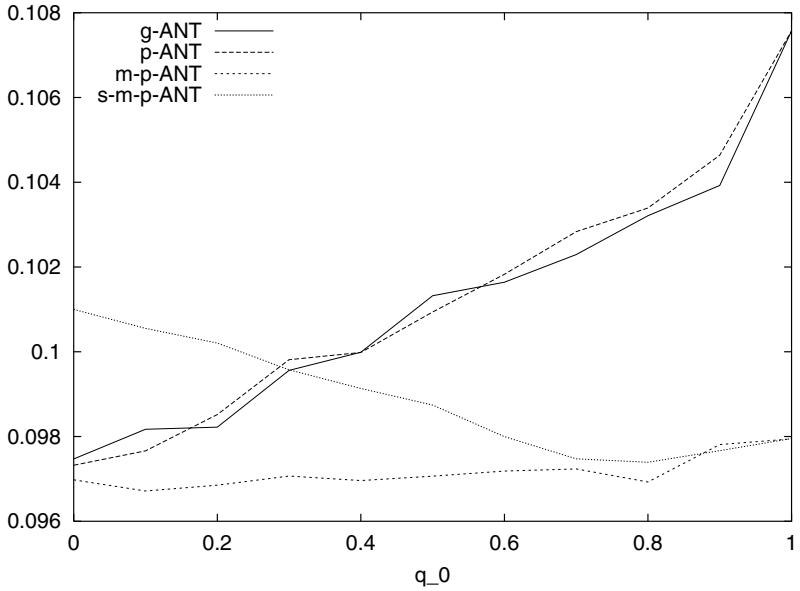

Figure 5. Solution quality (divided by 10^6) of algorithms g-ANT, p-ANT, m-p-ANT, and s-m-p-ANT for QAP instance sko81 for different values of q_0.

$q_0 = 1$. For s-m-p-ANT we have a behavior that is opposite to p-ANT and g-ANT, that is, the higher the value of q_0 the better the solution quality is. For all three test instances the solution qualities that are obtained with the respective best q_0 values by g-ANT and s-m-p-ANT are quite similar.

A problem with small q_0 values is that the simple threshold rule (equation (4)) used by s-m-p-ANT handles all locations that have a high pheromone value in the same way. The Modified Pseudo-Random-Proportional Action Choice Rule always chooses the item with the highest pheromone value from those items that were selected. Thus with a high q_0 value a location has a better chance of being chosen in s-m-p-ANT the higher its pheromone value is. This forces the set of items with a high pheromone value to shrink, so that the algorithm will search more near the best solutions and finally converge. Figures 6 and 7 illustrate this. Figure 6 shows the effect of q_0 on the number of items (locations) with a high pheromone value (i.e. $>t$) for the first decision of an ant for s-m-p-ANT on instance sko81. It can be seen that the number of items remains higher (namely, 10 for $q_0 = 0$) until the end of the run the smaller q_0 is. Figure 7 shows the value of h/l for the first decision of s-m-p-ANT on QAP instance sko81. For high q_0 values the algorithm concentrates the search with high probability on the few items with a high pheromone value.

A possible advantage of the Modified Pseudo-Random-Proportional Action Choice Rule is that it does not concentrate too much on the item with the maximal pheromone value. This prevents the algorithm from convergence too early. An indication of this is that results for m-p-ANT and s-m-p-ANT of $\rho = 0.98$ are even slightly better than that of the standard algorithm g-ANT. For larger values of ρ this advantage might not be so important. Therefore, we tested the influence of ρ

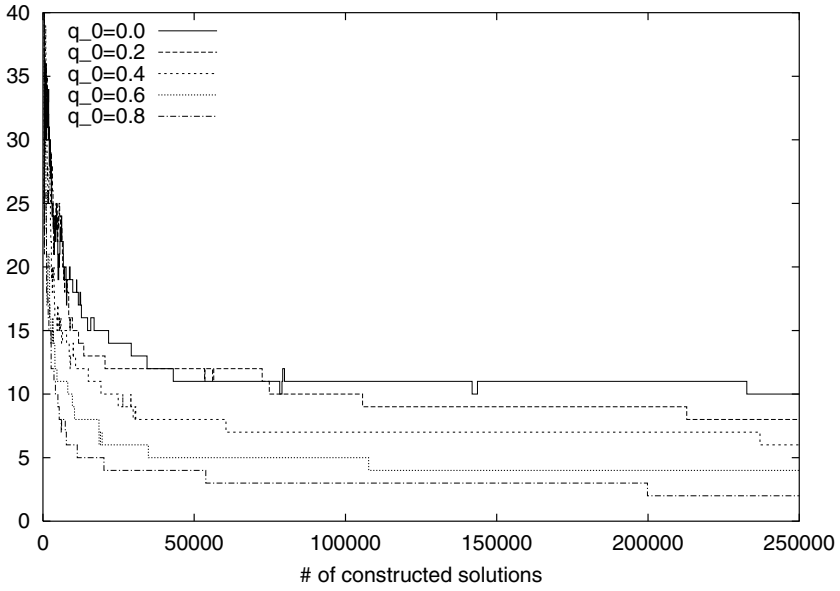

Figure 6. Number of items (locations) with high pheromone value ($>t$) for the first decision of a typical run of s-m-p-ANT with different values of q_0 on QAP instance sko81.

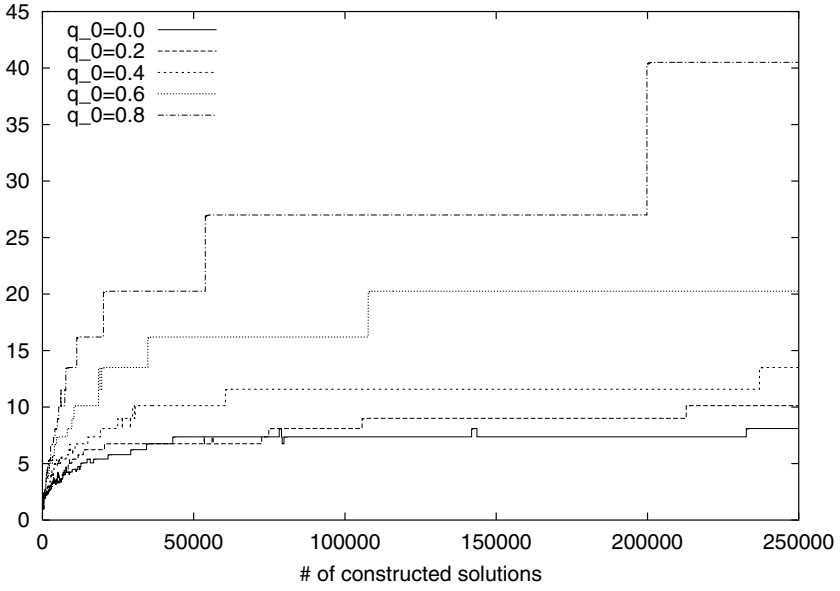

Figure 7. Value of h/l for the first decision of a typical run of s-m-p-ANT on QAP instance sko81.

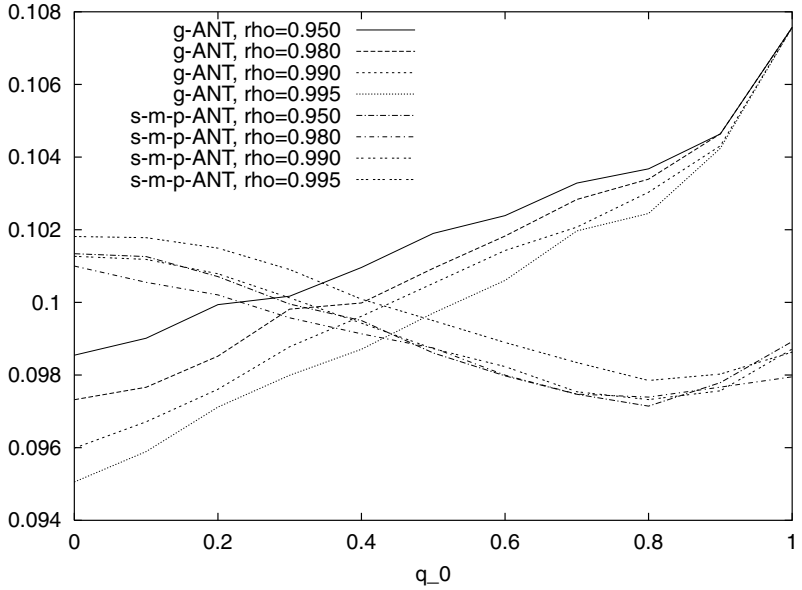

Figure 8. Solution quality (divided by 10^6) of algorithms g-ANT and s-m-p-ANT for different values of ρ.

on the solution quality (see Figure 8). The results of g-ANT improve with a growing value ρ. This effect is stronger for small values of q_0. In contrast to that, the quality of the solutions found by s-m-p-ANT does not depend much on the value of ρ. Hence, for medium and small values of ρ, algorithm s-m-p-ANT performs equally good or even slightly better than g-ANT, but for large values of ρ, g-ANT is better. To be fair, it has to be mentioned that g-ANT needs to construct, on the average, a smaller number of solutions as s-m-p-ANT, until it finds its best solution.

Recall, that the number of ants that are allowed to update in s-m-p-ANT depends on randomness. On the average every 10.02th ant was allowed to update in a typical run on sko81. This shows that the average updating behavior of s-m-p-ANT is quite similar to that of algorithm g-ANT.

The experiments have shown promising results for s-m-p-ANT. The reader should observe that it is still possible to improve s-m-p-ANT in several ways. For example, instead of using only two values h and l when computing the selection probabilities, any constant number can be used without changing the asymptotic running time, (with respect to Big-O) of s-m-p-ANT. Also, the use of other threshold values might improve the results. It will also be interesting to study for p-ANT, how the optimization behavior changes when the solution of each ant is compared with the solutions of the x preceding ants and the y following ants for different values of x and y (here we tried only $x = \lceil (m-1)/2 \rceil$ and $y = \lfloor (m-1)/2 \rfloor$).

8. Conclusion

We have proposed a new ACO algorithm called s-m-p-Ant that is suitable for fast parallel implementation. It was shown that s-m-p-Ant can be implemented in

quasilinear time (with respect to problem size n and the number of ants) on a reconfigurable mesh. Experiments on instances of the QAP have shown that the algorithm has good optimization behavior which is only slightly worse when compared to a standard ant algorithm. It will be interesting to study the optimization behavior of the algorithm for other permutation scheduling problems. Moreover, the abandoning of the generational principle deserves further study for sequential implementations. Future work would be to find fast parallel implementations for other ACO algorithms that use a different pheromone encoding.

Acknowledgment

We thank the reviewers for their helpful comments.

Note

1. $\log^* n$ is the number of log-operations that have to be applied to n until the result is at most 1; for all realistic n the value of $\log^* n$ is at most 6.

References

1. A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss, "Minimizing total tardiness on a single machine using ant colony optimization," Central European Journal of Operations Research, vol. 8, pp. 125–141, 2000.
2. M. L den Besten, T. Stützle, and M. Dorigo, "Ant colony optimization for the total weighted tardiness problem," in Parallel Problem Solving from Nature: 6th International Conference, M. Schoenauer et al. (eds.), Springer: Berlin, LNCS, vol. 1917, pp. 611–620, 2000.
3. K. Bondalapati and V. K. Prasanna, "Reconfigurable meshes: Theory and practice," in Proc. Reconfigurable Architectures Workshop, R. W. Hartenstein and Viktor K. Prasanna (eds.), April 1, 1997, Geneva, Switzerland, ITpress Verlag: Bruchsal.
4. B. Bullnheimer, G. Kotsis, and C. Strauss, "Parallelization strategies for the ant system," in High Performance Algorithms and Software in Nonlinear Optimization, R. De Leone, A. Murli, P. Pardalos, and G. Toraldo (eds.), Series: Applied Optimization, Kluwer: Dordrecht, vol. 24, pp. 87–100, 1998.
5. R. K. Congram, C. N. Potts, and S. L. van de Velde, "An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem," Working Paper, University of Southampton, UK, 1998.
6. M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic," in New Ideas in Optimization, D. Corne, M. Dorigo, F. Glover (eds.), McGraw-Hill: London, pp. 11–32, 1999.
7. M. Dorigo and L. M. Gambardella, "Ant colonies for the QAP," Technical Report IDSIA-4-97, IDSIA, Lugano, 1997.
8. M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," IEEE Transactions on Evolutionary Computation, vol. 1, pp. 53–66, 1997.
9. L. M. Gambardella, E. Taillard, and M. Dorigo, "Ant colonies for the quadratic assignment problem," Journal of the Operational Research Society, vol. 50, pp. 167–176, 1999.
10. S. Hartmann, and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," European Journal of Operational Research, vol. 127, pp. 394–407, 2000.

11. V. Maniezzo, "Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem," INFORMS Journal on Computing, vol. 11, pp. 358–368, 1999.
12. V. Maniezzo, A. Colorni, and M. Dorigo, "The ant system applied to the quadratic assignment problem," Tech. Rep. IRIDIA/94-28, Université Libre de Bruxelles, Belgium, 1994.
13. V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," IEEE Transactions on Knowledge and Data Engineering, vol. 11, pp. 769–778, 1999.
14. D. Merkle and M. Middendorf, "An ant algorithm with global pheromone evaluation for scheduling a single machine," Applied Intelligence, to appear.
15. D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," in Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2000), D. Whitley et al. (eds.), Morgan Kaufmann, pp. 893–900, 2000.
16. N. Meuleau and M. Dorigo, "Ant Colony Optimization and Stochastic Gradient Descent," Tech. Rep. IRIDIA/2000-36, Université Libre de Bruxelles, Belgium, 2000.
17. R. Michels and M. Middendorf, "An ant system for the shortest common supersequence problem," in New Ideas in Optimization, D. Corne, M. Dorigo, and F. Glover (eds.) McGraw-Hill, pp. 51–61, 1999.
18. M. Middendorf, "Bit-summation on the reconfigurable mesh," in Parallel and Distributed Computing, Proc. of the 11 IPPS/SPDP'99 Workshops, 6th Reconfigurable Architectures Workshop RAW-99, J. Rolim et al. (eds.), Springer: Berlin, LNCS, vol. 1586, pp. 625–633, 1999.
19. M. Middendorf, F. Reischle, and H. Schmeck, "Multi colony ant algorithms," Journal of Heuristics, vol. 8, pp. 305–320, 2002.
20. R. Miller, V. K. Prasanna-Kumar, D. I. Reisis, and Q. F. Stout, "Parallel computations on reconfigurable meshes," IEEE Trans. Comput., vol. 42, pp. 678–692, 1993.
21. K. Nakano and K. Wada, "Integer summing algorithms on reconfigurable meshes," Theoret. Comput. Sci., vol. 197, pp. 57–77, 1998.
22. T. Stützle, "An ant approach for the flow shop problem," in Proc. 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT '98), Verlag Mainz: Aachen, vol. 3, pp. 1560–1564, 1998.
23. T. Stützle and M. Dorigo, "ACO algorithms for the quadratic assignment problem," in New Ideas in Optimization, D. Corne, M. Dorigo, and F. Glover (eds.), McGraw-Hill, pp. 33–50, 1999.
24. T. Stützle and H. H. Hoos, "The MAX-MIN ant system," Future Generation Computer Systems, vol. 16, pp. 889–914, 2000.
25. http://mscmga.ms.ic.ac.uk/jeb/orlib/wtinfo.html.
26. http://www.bwl.uni-kiel.de /Prod/psplib/index.html.