Ant Colony Optimization for Resource-Constrained Project Scheduling

Daniel Merkle, Martin Middendorf, and Hartmut Schmeck

***Abstract*—An ant colony optimization (ACO) approach for the resource-constrained project scheduling problem (RCPSP) is presented. Several new features that are interesting for ACO in general are proposed and evaluated. In particular, the use of a combination of two pheromone evaluation methods by the ants to find new solutions, a change of the influence of the heuristic on the decisions of the ants during the run of the algorithm, and the option that an elitist ant forgets the best-found solution are studied. We tested the ACO algorithm on a set of large benchmark problems from the Project Scheduling Library. Compared to several other heuristics for the RCPSP, including genetic algorithms, simulated annealing, tabu search, and different sampling methods our algorithm performed best on average. For nearly one-third of all benchmark problems, which were not known to be solved optimally before, the algorithm was able to find new best solutions.**

***Index Terms*—Ant algorithms, ant colony optimization, metaheuristics, project scheduling, RCPSP, summation evaluation.**

I. Introduction

THE RESOURCE-constrained project scheduling problem (RCPSP) is a general scheduling problem that contains the job-shop, flow-shop, and open-shop problems as special cases. The RCPSP has attracted many researchers during the last years (see [4], [17] for recent overviews). Since it is an NP-hard problem, different kinds of heuristics have been proposed. A comparison of various heuristics on a set of benchmark problems is given in [16]. The compared heuristics include priority-based methods (e.g., multipriority rules, forward-backward scheduling, sampling methods) that use different serial schemes or parallel schedule generation schemes (PSGSs) and allow iterative searches by biasing the selection of the priority rule through a random device. Moreover, several metaheuristics as genetic algorithms, simulated annealing, and tabu search have been tested.

In this paper, we propose an ant colony optimization (ACO) approach for the RCPSP (see [9] for an introduction to ACO). The ACO approach has been applied recently to scheduling problems, as job-shop, flow-shop, and single machine tardiness problems (see [1], [5], [7], [18], [28], [29], [35], [39]). In ACO, several generations of artificial ants search for good solutions. Every ant of a generation builds a solution step by step going through several probabilistic decisions. In general, ants that find 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 generations are attracted by the pheromone so that they search in the solution space near previous good solutions. In addition to the pheromone values, the ants will usually be guided by some problem-specific heuristic for evaluating the possible decisions.

The algorithms proposed in [1] and [35] for the single machine total tardiness problem and the flow-shop problem, respectively, use a pheromone matrix (τ_{ij}) where pheromone is added to an element τ_{ij} of the pheromone matrix when a good solution was found where job j is the ith job on the machine. The following ants of the next generation directly use the value of τ_{ij} to estimate the desirability of placing job j as the ith job on the machine when computing a new solution. We call this use of the pheromone values direct or local evaluation (of the values in the pheromone matrix).

A different way to evaluate the pheromone matrix was proposed in [28]. Instead of using only the (local) value of τ_{ij} to derive the probability for placing job j as the ith on the machine it was proposed to give the ants a more global view of the pheromone values. In the particular form of global pheromone evaluation used in [28] for the single machine total tardiness problem, the ants evaluate the pheromone matrix by using $\sum_{k=1}^{i} \tau_{kj}$ to compute the probability of placing job j as the ith on the machine. We call this evaluation method summation evaluation (of the values in the pheromone matrix). A difficulty when using direct evaluation occurs if an ant does not choose job j as the ith job in the schedule even if τ_{ij} has a high value. When in this case the values $\tau_{i+1,j}, \tau_{i+2,j}, \ldots$ are small, job j is scheduled with high probability much behind place i. This is bad for many scheduling problems and, in particular, when the tardiness of jobs is to be minimized or when precedence constraints might hinder other jobs to be scheduled. With the summation evaluation method, this will usually not happen, since it is unlikely that a job is placed much behind places in the schedule that have corresponding high pheromone values.

We propose a combination of direct and summation evaluation for solving the RCPSP. We use τ_{ij}, respectively, $\sum_{k=1}^{i} \tau_{kj}$, to compute the probability that activity j is the ith activity that is scheduled by a serial schedule generation scheme (SSGS). Moreover, we propose several new features that are interesting for ACO algorithms in general and show that they are important to obtain a rather competitive algorithm for the RCPSP. One such feature is the changing influence of a deterministic heuristic on the decisions of the ants. During the initial generations, a high influence of the heuristic is important to guide the

Manuscript received February 16, 2001; revised September 4, 2001.

D. Merkle and H. Schmeck are with the Institute of Applied Informatics and Formal Description Methods, University of Karlsruhe, D-76128 Karlsruhe, Germany (e-mail: merkle@aifb.uni-karlsruhe.de; schmeck@aifb.uni-karlsruhe.de).

M. Middendorf is with the Computer Science Group, Catholic University of Eichstätt-Ingolstadt, D-85072 Eichstätt, Germany (e-mail: martin.middendorf@ku-eichstaett.de).

Publisher Item Identifier 10.1109/TEVC.2002.802450.

ants while in later generations the deterministic heuristic should not hinder the ants to find better solutions. Another feature concerns the use of an elitist ant. Usually in ACO, an elitist ant is an ant that stores the best solution found so far and is allowed to increase the corresponding pheromone values in every generation. This will lead the ants to search in the neighborhood to the best-so-far solution. Here, we propose to limit the influence of each elitist solution by replacing it with the best-found solution in the current generation after a certain number of generations.

The paper is organized as follows. The RCPSP is defined in Section II. In Section III, we describe the SSGSs. Our basic ACO algorithm is described in Section IV. The different deterministic heuristics we use in the ACO algorithm are described in Section V. Additional features of the ACO algorithm are presented in Section VI. The benchmark problems that were used for our tests and the parameter settings of the algorithms are described in Section VII. Experimental results are reported in Section VIII. Conclusions are given in Section IX.

II. Resource-Constrained Scheduling Problem

The RCPSP is the optimization problem to schedule the activities of a project such that the makespan of the schedule is minimized while given precedence constraints between the activities are satisfied and resource requirements of the scheduled activities per time unit do not exceed given capacity constraints for the different types of resources.

Formally, $\mathcal{J} = \{0, \ldots, n+1\}$ denotes the set of activities of a project. We assume that a precedence relation is given between the activities. An activity list is a permutation of the activities such that every activity comes in the list before all activities that are its successors in the precedence relation. $\mathcal{Q}$ is a set of q resource types. $R_i > 0$ is the resource capacity for resources of type $i \in \mathcal{Q}$. Every activity $j \in \mathcal{J}$ has a duration p_j and resource requirements $r_{j,1}, \ldots, r_{j,q}$, where $r_{j,i}$ is the requirement for a resource of type i per time unit when activity j is scheduled.

Let $\mathcal{P}_j(\mathcal{S}_j)$ be the set of direct predecessors (respectively successors) of activity j. $\mathcal{P}_j^*(\mathcal{S}_j^*)$ is the set of all predecessors (respectively successors) of activity j. Activity 0 is the only start activity that has no predecessor and activity $n+1$ is the only end activity that has no successor. We assume that the start activity and the end activity have no resource requirements and have duration zero.

A schedule for the project is represented by the vector $(s_0, s_1, \ldots, s_{n+1})$, where s_j is the start time of activity $j \in \mathcal{J}$. If s_j is the start time of activity j, then $f_j = s_j + p_j$ is its finish time. For a schedule, the start time is the minimum start time $\min\{s_j \mid j \in \mathcal{J}\}$ of all activities, and the finish time is the maximum finish time $\max\{s_j + p_j \mid j \in \mathcal{J}\}$ of all activities. The makespan of a schedule is the difference between its finish time and start time. Observe that the start time of a schedule equals s_0 and the finish time equals f_{n+1}.

A schedule is feasible if it satisfies the following constraints: 1) activity $j \in \mathcal{J}$ must not be started before all its predecessors are finished, i.e., $s_j \geq s_i + p_i$ for every $i \in \mathcal{P}_j$, and 2) the resource constraints have to be satisfied, i.e., for every time unit t, the sum of the resource requirements of all scheduled activities does not exceed the resource capacities. Formally, for every time unit t and every resource of type i, it holds that $\sum_{j \in \mathcal{J}, s_j \leq t < s_j + p_j} r_{j,i} \leq R_i$.

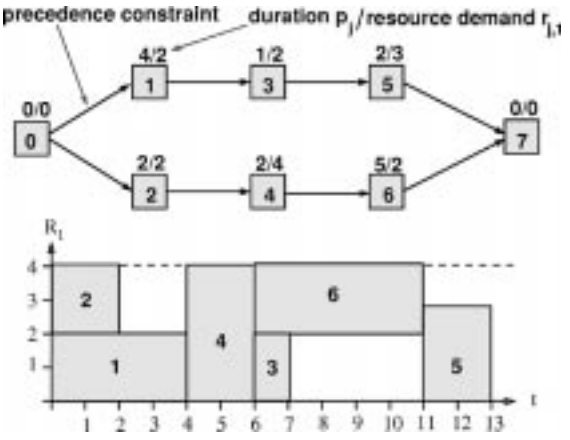

Fig. 1. Precedence graph for an RCPSP instance with activities $0, \ldots, 7$, one resource with capacity 4 and a corresponding feasible schedule.

The RCPSP problem is the following: given a project with resource constraints, find a feasible schedule with minimal makespan. See Fig. 1 for an example of an RCPSP instance with a corresponding feasible schedule.

A latest start time LS_j and a latest finish time LF_j can be computed for an activity j by backward recursion from an upper bound of the finish time T of the project (cf. [13]). Starting with $LS_{n+1} = LF_{n+1} = T$, define $LF_j = \min\{LS_i \mid i \in \mathcal{S}_j\}$ and $LS_j = LF_j - p_j$ for $j = n, \ldots, 0$. An earliest start time ES_j and an earliest finish time EF_j can be computed for every job j in a forward pass as follows. Starting with $ES_0 = EF_0 = 0$, define $ES_j = \max\{EF_i \mid i \in \mathcal{P}_j\}$ and $EF_j = ES_j + p_j$, for $j = 1, \ldots, n+1$. Note, that the earliest start time ES_{n+1} of job $n+1$ is a trivial lower bound on the minimal makespan of a feasible schedule.

In the following, we list some notations that are used in this paper (and are defined in the following sections).

RCPSP:

$\mathcal{J}$	Set of activities.
$\mathcal{Q}$	Set of resource types.
R_i	Capacitiy of resource type i.
$r_{j,i}$	Requirement of activity j for resource type i.
p_j	Duration of activity j.
$\mathcal{P}_j(\mathcal{S}_j)$	Set of direct predecessors (respectively successors) of activity j.
$\mathcal{P}_j^*(\mathcal{S}_j^*)$	Set of all predecessors (respectively successors) of activity j.

Schedule:

s_j	Start time of activity j.
f_j	Finish time of activity j.
F_j	Maximum finish time of immediate predecessors of activity j.
$LS_j(ES_j)$	Latest (earliest) start time of activity j.
$LF_j(EF_j)$	Latest (earliest) finish time of activity j.

ACO Algorithm:

τ_{ij}	Pheromone information.
η_{ij}	Heuristic information.

c	Relative influence of direct evaluation.
$\alpha(\beta)$	Influence of pheromone (heuristic) information.
γ	Influence of earlier decisions.
ρ	Evaporation rate.
$g_{\max}$	Maximal number of generations to hold an elitist solution.
$\mathcal{E}$	Set of eligible activities.
T^*	Makespan of best-found schedule.

III. Schedule Generation Scheme

We used two different serial scheduling schemes for the generation of a schedule. Both schedule generation schemes, SSGS and PSGS, are standard heuristic methods for the RCPSP (cf. [24]).

The SSGS starts with a partial schedule that contains only the start activity 0 at time 0. Then, SSGS constructs the complete schedule in n stages, where at each stage one activity is added to the partial schedule constructed so far. In every stage g, one activity j is selected from the set of available activities $\mathcal{E}(g)$, i.e., activities that have not been scheduled so far and where each predecessor has already been scheduled. The SSGS can be supplied with an activity list, i.e., an ordered list of all activities. In this case, SSGS always chooses the first activity in the list that is feasible.

The start time of the activities are determined by SSGS as follows. For every eligible activity $j \in \mathcal{E}(g)$, let F_j be the maximum finish time of all its immediate predecessors. Then, the start time of activity j is the earliest time in $[F_j, LF_j - p_j]$ such that all resource constraints are satisfied (see Procedure 1).

Procedure 1 Serial Schedule Generation Scheme
$\mathcal{E}(g)$: all activities which can be started precedence-feasible in stage g
1: **for** $g = 0$ to $n+1$ **do**
2: Calculate the eligible set $\mathcal{E}(g)$
3: Select one $j \in \mathcal{E}(g)$
4: Schedule j at the earliest precedence- and resource-feasible start time $t \in [F_j, LS_j]$
5: **end for**

The PSGS proceeds by time incrementation (see Procedure II). It starts with a partial schedule that contains only the start activity 0 at time 0. In every stage g, a start time t_g for the next activities is determined before these activities are actually chosen. The set of all eligible activities $\mathcal{E}(t_g)$, which are precedence- and resource-feasible when scheduled at time t_g, is computed. The following selection steps are done until $\mathcal{E}(t_g)$ is empty. An activity j is selected from $\mathcal{E}(t_g)$ and scheduled at time t_g. Then, $\mathcal{E}(t_g)$ is recomputed. At the end of the stage, a new start time t_{g+1} for the next activities is chosen. Similarly as for the SSGS, the PSGS can be supplied with an activity list. In this case, PSGS choses from $\mathcal{E}(t_g)$ always the activity that comes first in the activity list.

Procedure 2 Parallel Schedule Generation Scheme
$\mathcal{E}(t_g)$: all activities which are precedence- and resource-feasible when started at t_g,
$\mathcal{C}$: set of already scheduled activities,
active set $\mathcal{A}_g = \{j \in \mathcal{C} | s_j \leq t_g < f_j\}$
1: $g := 0$, $t_g := 0$, $\mathcal{C} := \{0\}$
2: **while** $|\mathcal{J} - \mathcal{C}| > 0$ **do**
3: Calculate the eligible set $\mathcal{E}(t_g)$
4: **while** $\mathcal{E}(t_g) \neq \emptyset$ **do**
5: Select one $j \in \mathcal{E}(t_g)$
6: Schedule j at t_g
7: $\mathcal{C} := \mathcal{C} \cup \{j\}$
8: Recalculate $\mathcal{E}(t_g)$
9: **end while**
10: Calculate $\mathcal{A}_g$
11: $g := g + 1$
12: Calculate the minimal finish time t_g of all activities in $\mathcal{A}_{g-1}$
13: **end while**

It is known that for every RCPSP instance, it is possible to obtain an optimal solution by using the SSGS. That means there exists a sequence of choices that can be made by the SSGS and leads to an optimal schedule (e.g., [24]). The PSGS always yields nondelay schedules, i.e., schedules where idle times are introduced only when no feasible activity is available, but these do not necessarily include an optimal solution.

IV. ACO Algorithm

The general idea of our ACO approach is to use the ant algorithm for finding an activity list that gives a good schedule when used by SSGS or PSGS. The principle of our ACO algorithm is similar to an ACO algorithm called Ant System Traveling Salesperson Problem (AS-TSP) for the TSP of [8] and [11]. In every generation, each of m ants constructs one solution. An ant selects the activities in the order in which they will be used by the SSGS or PSGS. For the selection of an activity the 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 put activity j at place i of the activity list for the SSGS or PSGS. The heuristic value is generated by some problem-dependent heuristic and the pheromone information stems from former ants that have found good solutions.

The next activity is chosen according to the probability distribution over the set of eligible activities $\mathcal{E}$ determined either by direct evaluation [8], [11] according to

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} [\eta_{ij}]^{\beta}}{\sum_{h \in \mathcal{E}} [\tau_{ih}]^{\alpha} [\eta_{ih}]^{\beta}} \tag{1}$$

or by summation evaluation [28] according to

$$p'_{ij} = \frac{\left(\sum_{k=1}^{i} \left[\gamma^{i-k} \cdot \tau_{kj} \right] \right)^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{h \in \mathcal{E}} \left(\sum_{k=1}^{i} \left[\gamma^{i-k} \cdot \tau_{kh} \right] \right)^{\alpha} \cdot [\eta_{ih}]^{\beta}} \tag{2}$$

where parameter $\gamma > 0$ determines the relative influence of pheromone values corresponding to earlier decisions, i.e.,

preceding places in the permutation. A value $\gamma = 1$ results in unweighted summation evaluation, i.e., every τ_{kj}, $k \leq i$ is given the same influence. A value $\gamma < 1$ ($\gamma > 1$) gives pheromone values corresponding to earlier decisions less (respectively more) influence. Parameters α and β determine the relative influence of the pheromone values and the heuristic values on the decision of the ant. The different heuristics that have been used in this paper are described in the next section.

The best solution found so far and the best solution found in the current generation are used to update the pheromone information. However, before that, some portion of pheromone is evaporated according to

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

where ρ is the evaporation rate. The reason for this is that old pheromone should not have too strong an influence on the future. Then, for every activity $j \in \mathcal{J}$, some amount of pheromone is added to element τ_{ij} of the pheromone matrix, where i is the place of activity j in the activity list of the best solution found so far. This is an elitist strategy that leads ants to search near the best-found solution. The amount of pheromone added is $\rho/2T^*$, where T^* is the makespan of the best-found schedule, i.e.,

$$\tau_{ij} = \tau_{ij} + \rho \cdot \frac{1}{2T^*}.$$

The same is done also for the best solution found in the current generation, i.e., for every activity $j \in \mathcal{J}$, pheromone is added to τ_{ij} when i is the place of activity j in the activity list of the best solution found in the current generation. The algorithm runs until some stopping criterion is met, e.g., a certain number of generations has been done or the average quality of the solutions found by the ants of a generation has not changed for several generations.

V. Heuristics

As heuristics, we use adaptations of well-known priority heuristics for the RCPSP (see [19] for an overview). The reason to use adaptations is that for the decisions of the ants, the relative heuristic values are important and not just their ranks. The importance of this aspect has been pointed out before in [28].

The latest finish time (LFT) heuristic [6] schedules activities according to growing values of LF. The relative differences between the latest finish times are usually small for activities that become eligible late. Therefore, we use the absolute differences to the maximum latest finish time of an eligible activity as a heuristic value. In particular, for the normalized LFT (nLFT) heuristic, the values η_{ij} are computed for the eligible activities according to

$$\eta_{ij} = \max_{k \in \mathcal{E}} LF_k - LF_j + 1.$$

The normalized version of the latest start time (LST) called normalized LST (nLST) heuristic is defined similarly to the nLFT heuristic, but uses LS values instead of LF values. Therefore, for the nLST heuristic, we define

$$\eta_{ij} = \max_{k \in \mathcal{E}} LS_k - LS_j + 1.$$

For the normalized version of the minimum slack time (MSL), called normalized MSL (nMSL) heuristic, we define

$$\eta_{ij} = \max_{k \in \mathcal{E}} (LS_k - ES_k) - (LS_j - ES_j) + 1.$$

The most total successors (MTS) heuristic always prefers the activity with the largest number of (not only direct) successor activities. For the normalized MTS (nMTS), we define

$$\eta_{ij} = |\mathcal{S}_j^*| - \min_{k \in \mathcal{E}} |\mathcal{S}_k^*| + 1.$$

The greatest rank positional weight all (GRPWA) heuristic always prefers the activity with the largest sum of durations of all (not only direct) successor activities plus the duration of the activity itself. For the normalized GRPWA (nGRPWA), we define

$$\eta_{ij} = p_j + \sum_{i \in \mathcal{S}_j^*} p_i - \min_{k \in \mathcal{E}} \left(p_k + \sum_{i \in \mathcal{S}_k^*} p_i \right) + 1.$$

The weighted resource utilization and precedence (WRUP) heuristic [37] considers the resource utilization and the number of direct successors of an activity. For the normalized version, called normalized WRUP (nWRUP) heuristic, we define

$$\eta_{ij} = \omega |\mathcal{S}_j| + (1 - \omega) \sum_{l \in \mathcal{Q}} \frac{r_{jl}}{R_l} - \min_{k \in \mathcal{E}} \left(\omega |\mathcal{S}_k| + (1 - \omega) \sum_{l \in \mathcal{Q}} \frac{r_{kl}}{R_l} \right) + 1$$

where $\omega \in [0, 1]$ is a parameter of the heuristic.

VI. Additional Features

In this section, we describe several additional features of our ACO algorithm.

A. Combination of Direct and Summation Evaluation

Summation evaluation was introduced in [28] and applied to the single machine total weighted tardiness problem. For this problem it is important to schedule jobs not too late, i.e., not later than their tardiness value, which is exactly what the summation evaluation enforces. For the RCPSP the situation is somewhat different. The precedence relation requires that some activities should be scheduled not too late, but it is also important to schedule groups of activities at the same time that have resource requirements fitting to the resource constraints of the problem. Therefore, for some activities, there might be several places in the activity list used by SSGS or PSGS that are good, while other places in between might be worse. Such a behavior can be modeled with direct evaluation rather than using summation evaluation. Therefore, instead of using either pure direct evaluation or pure summation evaluation we propose a combination of both evaluation strategies for the RCPSP. Such a combination is obtained as follows. A parameter c, $0 \leq c \leq 1$ determines the relative influence of direct evaluation and summation evaluation. The probability distribution used by an ant for choosing

the next activity is computed as in formula (1) but the values τ_{ij} are replaced by the following "new" values τ'_{ij}

$$\tau'_{ij} := c \cdot x_i \cdot \tau_{ij} + (1-c) \cdot y_i \cdot \sum_{k=1}^{i} \gamma^{i-k} \tau_{kj}$$

where $x_i := \sum_{h \in \mathcal{E}} \sum_{k=1}^{i} \gamma^{i-k} \tau_{kh}$ and $y_i := \sum_{h \in \mathcal{E}} \tau_{ih}$ are factors to adjust the relative influence of direct and summation evaluation. Observe that for $c = 1$, we obtain pure direct evaluation and for $c = 0$ pure summation evaluation.

B. Changing Parameter Values

Usually the main parameters of an ACO algorithm, i.e., α, β, ρ, are assigned fixed values during the whole run of the algorithm. Several authors discussed the problem of finding good parameter values and have studied the influence of the problem instance on the optimal parameter values. For example, in [2], a genetic algorithm was used to find good values for parameters α and β. For an approach to use ant algorithms for multiobjective optimization, it was shown in [30] that the use of different parameter values for the ants in the same generation can be advantageous. In all these works, no variation of parameter values over the generations is considered.

Here, we propose to change the values of parameters β and ρ in a certain way during a run of the algorithm. Parameter β controls the relative influence of the heuristic values. Usually, the heuristic values will help the first generations of ants finding good solutions but later on they might hinder the ants to follow the good pheromone trails and therefore make it difficult for the algorithm to further improve the solution quality it has found so far. This is especially a problem when a static heuristic is used, i.e., a heuristic where the values remain the same during the run of the algorithm. Therefore, we reduce the value of β from generation to generation until it becomes zero and only the pheromone values guide the decisions of the ants.

Parameter ρ determines the convergence speed of the algorithm. In general, when the algorithm has time to generate a large number of solutions, a low value of ρ is profitable since the algorithm will explore different regions of the search space and does not focus the search too early on a small region. However, when the maximal number of solutions is restricted, a higher value of ρ usually performs better. For this reason, we propose to use two different values for ρ during the run of an ant algorithm. A low ρ should be used from the start of the algorithm during most of the running time. Only for the last few generations, a high ρ should be used to make a final intensive search near the best solution that has been found so far.

C. Discarding the Elitist Solution

Usually, using an elitist strategy has the advantage that the ants explore the search region around the best-so-far solution intensively. A best-so-far solution that was stable for many generations has a great influence on the pheromone values since pheromone is added after every generation according to the best-so-far solution, but then, during long runs, it can happen that the algorithm converges too early to the best-found solution. Especially, if the elitist solution is a single good solution that has no other good solutions in its neighborhood, it is a problem when the search concentrates too much on this region.

We, therefore, set a maximal number $g_{\max}$ of generations during which an elitist solution is allowed not to change. When an elitist solution has exceeded its maximal number of generations $g_{\max}$ it is exchanged with the best solution in that generation, even if this solution is worse than the (old) elitist solution. When the old elitist solution has good solutions in its neighborhood it is likely that one of these good solutions is found soon by the ants and might become the new elitist solution. Otherwise, it does not matter that the old elitist solution has been forgotten.

A somewhat similar strategy in [36] was to switch between generations where the elitist ant updates pheromone according to the best solution found so far with generations where the update is done according to the best solution in that generation.

D. Local Optimization Strategy

It has been shown that local search applied to the solutions that the ants have found can improve the optimization behavior of an ACO algorithm [7], [10]. Here, we study the influence of two different local optimization strategies.

A local search strategy that was used in [15] in connection with a genetic algorithm for the RCPSP performs right moves of activities in the activity list. The activity list is used by the SSGS to build a schedule. Several criteria are identified in [15] under which a right move will not change the schedule. Only when all these criteria are not satisfied is the right move evaluated by creating the corresponding schedule.

A problem with the right-move strategy might be that solutions that can be obtained by a right move are likely to be found by the ants anyway. Therefore, we propose a 2-opt strategy that considers swaps between pairs of activities in a solution. For a pair i, j, $i < j$, of activities, it is checked whether or not the schedule derived by SSGS using the sequence, where i and j are exchanged is feasible and better than the schedule derived from the old sequence. If this is the case, the new sequence is fixed for testing the remaining swaps between pairs of activities.

E. Bidirectional Planning

The use of forward and backward ants that solve a problem instance from different directions was employed in [30] for the shortest common supersequence problem. In this paper, the same pheromone matrix was used by both kinds of ants concurrently. It was shown that the ACO algorithm could profit from the two types of ants, but to an extent, that depends on the type of the problem instance. A large profit was obtained for types of problem instances, where it is likely that the good solutions obtained by the forward ants are similar to the good solutions that were found by the backward ants.

For the RCPSP problem, such a property will usually not hold—at least this is unlikely when the construction of a solution is based on a SSGS. However, this does not necessarily mean that backward ants are useless for the RCPSP. The method to build up schedules from both sides of an instance is called bidirectional planning. It has been used before in connection with different heuristics for the RCPSP. In [12], bidirectional

planning was used in connection with a genetic algorithm. A special kind of tabu search algorithm for the RCPSP that swaps between phases where the tail of a solution is improved with phases where the head of a solution is improved has been investigated in [38]. The advantage of bidirectional planning in connection with deterministic priority rules was studied in [19].

In this paper, we use two colonies of ants—one colony consists of forward ants and the other of backward ants. Backward ants work on the reversed problem instances, i.e., the arcs of the precedence graph are reversed. Both colonies work separately on their own pheromone matrices. After a number of generations, both colonies compare their results, e.g., the average over the best solutions found during the last generations (to use the average instead of simply the best-so-far solution is proposed because it seems reasonable to base the decision in an early stage of the algorithm not only on a single solution). Then, only the colony with the better result continues with the optimization process.

VII. Benchmark Problems and Parameters

As benchmark problems, we used a set of test instances which is available in the Project Scheduling Library (PSPLIB) [25], [26]. From this library, we used the test set j120.sm, which contains the largest problem instances in the PSPLIB.

The benchmark problems in the set j120.sm were generated by varying the following three problem parameters: network complexity (NC), resource factor (RF), and resource strength (RS). NC defines the average number of predecessors per activity. RF determines the average percentage of different resource types for which each activity (besides the start and the end activity) has a nonzero-demand. RS defines how scarce the resources are. A value of 0 defines the capacity of each resource to be no more than the maximum demand of all activities while a value of 1 defines the capacity of each resource to be equal to the demand imposed by the earliest start time schedule.

The set j120.sm contains 600 problem instances, each having 120 activities and four resource types. The set contains ten instances for each combination of the following parameter values: $NC \in \{1.5, 1.8, 2.1\}$, $RF \in \{0.25, 0.5, 0.75, 1\}$, and $RS \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

When not mentioned explicitly, the parameter values of the ACO algorithm used for the test runs are chosen as follows. The number of ants per generation is $m = 5$. The number of ant generations per run of the algorithm is at most 850, during the first 100 generations we use two colonies of five ants (one with forward ants and one with backward ants, as described in Section VI-E). After the first 100 generations, the algorithm proceeds only with the colony that achieved on the average over the last 25 generations better best solutions. After 850 ant generations or when the average solution quality per generation did not change for ten generations, a local search phase is initiated. Local search is applied only to the best solution found (so far) by the algorithm until a total number of 5000 schedules (including all schedules found by the ants) have been evaluated (see Section VI-D). Thus, 250 local search steps are performed when all 850 generations of ants were carried out.

We set $\alpha = 1$, $c = 0.5$, and $\gamma = 1.0$ (unless stated otherwise). Recall that $c = 0.5$, $\gamma = 1.0$ means that local evaluation and summation evaluation have the same influence and for summation evaluation all pheromone values corresponding to the former decisions have the same influence. Parameter β was set to 2 in the first generation and is decreased linearly so that it becomes zero after 50% of the (maximal) number of generations, i.e., at generation 425. We used $\rho = 0.025$ and switched to a higher value of $\rho = 0.075$ for the last 200 generations of the algorithm. The elitist solution was discarded after every $g_{\max} = 10$ generations. The standard heuristic used by the ants was the nLST heuristic.

In the following, Ant System RCPSP (AS-RCPSP) denotes the algorithm with the parameters as described above (unless stated otherwise). We also tested a simpler version of the ACO algorithm that has none of the additional features as described in Section VI. In contrast to AS-RCPSP, this algorithm—simple AS-RCPSP (s-AS-RCPSP)—uses only forward ants, has constant values $\beta = 1$ and $\rho = 0.025$, and uses always 1000 generations of ants without local optimization at the end.

All tests have been performed on a Pentium III 500-MHz processor. One run of AS-RCPSP takes about 25 s for one problem instance.

The results for the test instances are compared to lower bounds (LB), which were obtained by a critical path heuristic [34]. Every test result given in the following section is an average over all 600 problem instances and over four runs for each instance (unless stated otherwise).

VIII. Experimental Results

A. s-AS-RCPSP

We studied the s-AS-RCPSP for different c and γ values using the SSGS and the PSGS. The combination between local evaluation and summation evaluation were tested with values $c \in \{0, 0.2, 0.4, 0.5, 0.6, 0.8, 1.0\}$ and $\gamma \in \{0, 0.2, 0.4, 0.6, 0.8, 0.9, 1.0, 1.1, 1.2, 1.4\}$. The results are given in Figs. 2 and 3. The results show that a combination of local evaluation and summation evaluation performed best for both SSGS and PSGS. The smallest deviations from the lower bounds were obtained for SSGS with 36.7% for $c = 0.6$ and $\gamma = 1$ and for PSGS with 37.6% for $c = 0.8$ and $\gamma = 1.1$. In both cases, summation evaluation has slightly more influence than local evaluation and all weights are close to 1 for summation evaluation. Nearly the worst performance was obtained in both cases by pure local evaluation ($c = 1.0$ or $\gamma = 0$) with a deviation of 40.3% for SSGS and 38.5% for PSGS. For PSGS, the worst results were obtained for $c = 0$ and $\gamma > 1.0$ with a deviation of 38.6%. Compared to that, summation evaluation performed much better when γ values were neither too large nor too small—for SSGS and $\gamma = 0.9$ the average deviation was only 37.1% and for PSGS and $\gamma \in \{0.4, 0.6, 0.8\}$ the average deviation was only 38.0%. While for s-AS-RCPSP with SSGS the best results were obtained, it seems that s-AS-RCPSP with PSGS is much more robust against changes of the c and γ parameters. The fact that with SSGS the best results were obtained for s-AS-RCPSP is in slight contrast to results given in [15] and [19]. The authors of these studies found

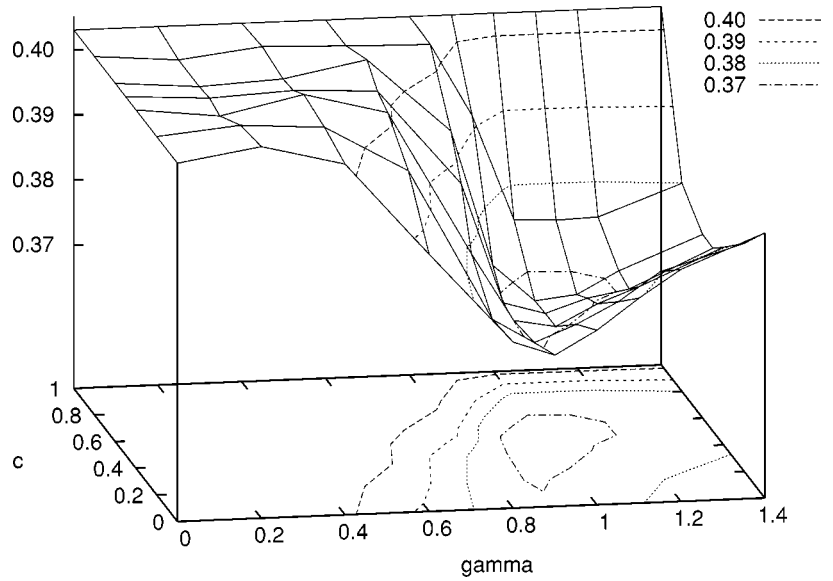

Fig. 2. Results for s-AS-RCPSP with SSGS with different values of c and γ. Shown is the deviation from the critical path lower bound.

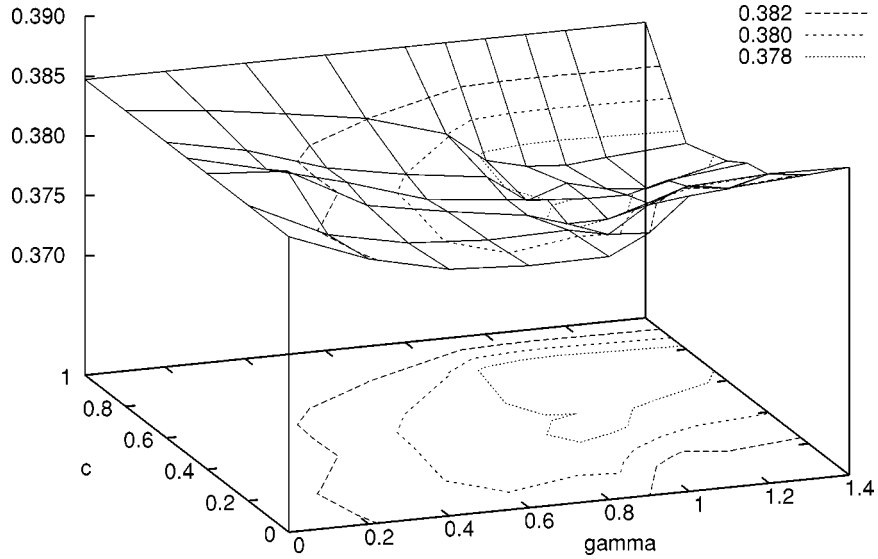

Fig. 3. Results for s-AS-RCPSP with PSGS with different values of c and γ.

that at least for large problem instances (as j120.sm) PSGS in combination with a GA or various deterministic heuristics found on the average better solutions. Since our best results were obtained with SSGS, we made the further investigations using s-AS-RCPSP with SSGS.

To study the influence of direct and summation evaluation on s-AS-RCPSP in more detail, we computed the entropy of the probability distributions over all eligible activities that are considered by the ants for choosing the next activity. That is, for every ith decision, $i \in [1,120]$, of an ant during the process of constructing a solution we computed the entropy

$$e_i = -\sum_{j \in \mathcal{E}} p_{ij} \log p_{ij}.$$

The entropy values show how much the algorithm has converged with respect to the different decisions of the ants.

Note that, in contrast to most other results in this paper, where only 5000 evaluations were allowed, we run the ant algorithm for 2000 generations (corresponding to 10 000 evaluations) to observe convergence of the algorithm. Figs. 4 and 5 show entropy curves for different generations of ants during a run. Each point of the curve is averaged over 300 values obtained from the five ants in the generation for 60 problem instances (we took the first instance of every problem type contained in j120.sm). One observation is that the entropy values corresponding to decisions in the middle of the construction process of a solution are larger than at the end or at the beginning of the process. A reason for this is that decisions in the middle have a larger set of eligible activities. Another observation is that for direct evaluation, the entropy values are much smaller than for the combination of direct and summation evaluation.

It is interesting that for pure direct evaluation $c = 1$, the entropy for the first decisions of an ant shrinks very fast, e.g., all

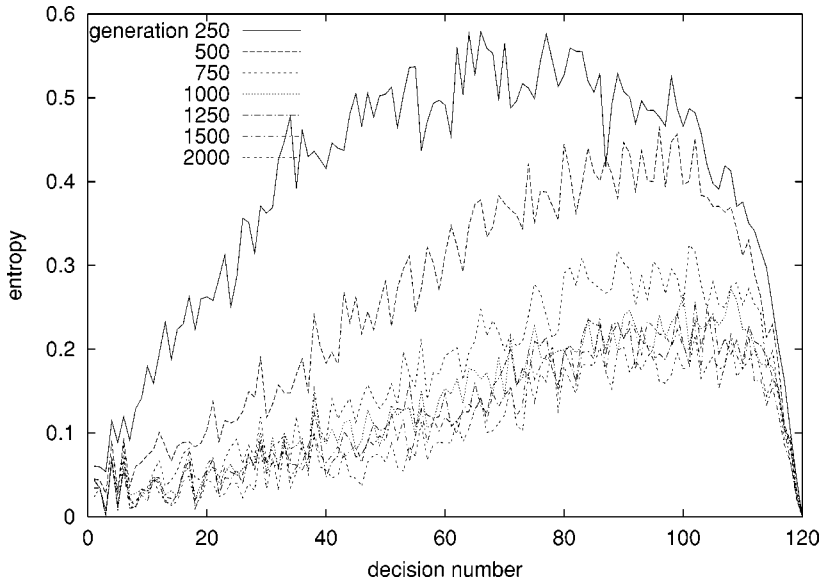

Fig. 4. Entropy of the probability distribution used by the ants for s-AS-RCPSP with $c = 1$. Results are averaged over five ants per generation for 60 problem instances. Note that the scales of Figs. 4 and 5 are different.

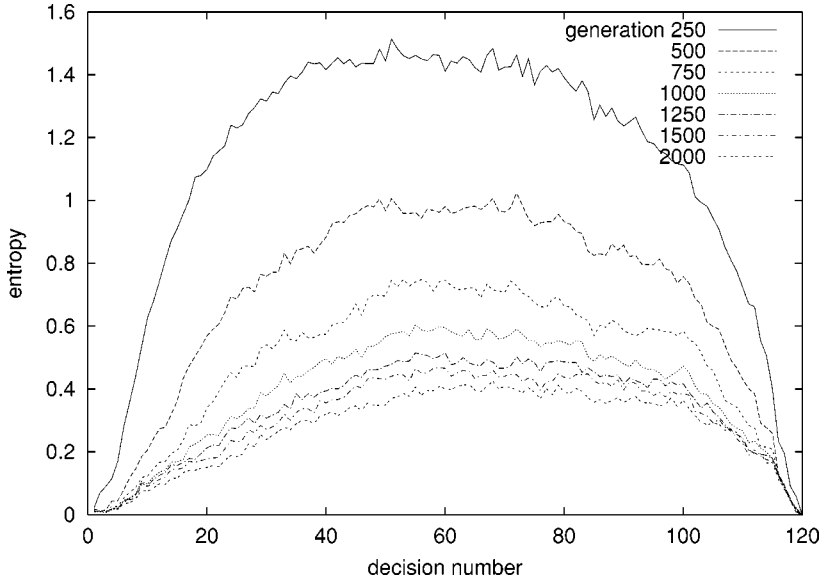

Fig. 5. Entropy of the probability distribution used by the ants for s-AS-RCPSP with $c = 0.5$. Results are averaged over five ants per generation for 60 problem instances. Note that the scales of Figs. 4 and 5 are different.

entropy values are below 0.2 for the first 60 decisions after generation 750. In contrast to that, the combination of direct and summation evaluation $c = 0.5$ shows for all generations nearly symmetrical curves, i.e., decisions at the beginning and at the end have similar entropy values. Also in this case, the entropy values are higher in later generations than for pure direct evaluation. This indicates an advantage of the combined evaluation method: it prevents the algorithm from nonuniform convergence and from too early convergence.

B. AS-RCPSP

The results of the enhanced AS-RCPSP algorithm with SSGS are shown in Fig. 6 and Table I. The general behavior is similar to the simple s-AS-RCPSP with SSGS. The best result was obtained for $c = 0.5$ and $\gamma = 1$ with a deviation of 35.43% from the critical path lower bound. The worst performance was obtained by pure local evaluation ($c = 1.0$ or $\gamma = 0$) with a deviation of 38.24%.

We compared AS-RCPSP to other heuristics for RCPSP. Several heuristics are included in an extensive experimental study by [16]. This study considers the following heuristics:

1) three deterministic single/pass heuristics with regret based random sampling from [21] and [22];
2) two single/pass heuristics with adaptive regret-based random sampling [23], [33];
3) four genetic algorithms of [14] and [27];
4) a simulated annealing algorithm of [3].

These heuristics were also compared with two pure random sampling methods using two different heuristics for building up a schedule. The random sampling 2 heuristic in Table II is a

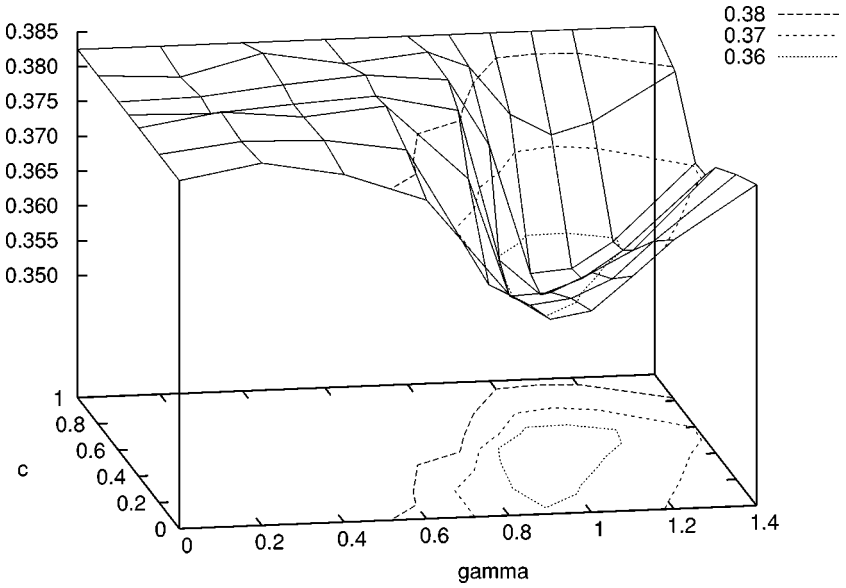

Fig. 6. Results for AS-RCPSP with different values of c and γ.

TABLE I
RESULTS FOR AS-RCPSP WITH DIFFERENT VALUES OF c AND γ

γ	c						
	0.0	0.2	0.4	0.5	0.6	0.8	1.0
0.0	38.2	38.2	38.2	38.2	38.2	38.2	38.2
0.2	38.4	38.4	38.4	38.2	38.3	38.2	38.2
0.4	38.2	38.4	38.3	38.3	38.4	38.8	38.2
0.6	37.8	38.2	38.4	38.4	38.5	38.8	38.2
0.8	36.4	36.2	37.3	38.1	38.4	38.8	38.2
0.9	**36.0**	**35.9**	**35.6**	35.9	37.4	38.6	38.2
1.0	36.1	**35.9**	**35.6**	**35.4**	**35.5**	38.0	38.2
1.1	36.6	36.2	35.9	35.6	35.6	**37.6**	38.2
1.2	37.1	36.7	36.2	36.0	36.0	37.9	38.2
1.4	37.9	37.6	37.3	37.0	37.0	38.4	38.2

Best results in every column are bold. The best result of the table is underlined.

TABLE II
COMPARISON OF AS-RCPSP WITH DIFFERENT RANDOMIZED HEURISTICS

Algorithm	**Reference**	**deviation from LB in %**
AS-RCPSP	this paper	35.43*
Self-adapting GA	[15]	35.60*
LS with LR-IP	[31]	(36.2)
GA 1	[14]	36.74*
TO B&B	[12]	(37.1)
SA	[3]	37.68*
GA 2	[14]	38.49
adaptive sampling 1	[33]	38.70
single pass/sampling 1	[22]	38.75
single pass/sampling 2	[21], [22]	38.77*
adaptive sampling 2	[23]	40.45*
GA 3	[27]	40.69*
single pass/sampling 3	[22]	41.84*
GA 4	[14]	42.25
random sampling 1	[20]	43.05*
random sampling 2	[20]	47.61

Every heuristic was allowed to construct and evaluate 5000 solutions. "$*$" indicates that the ranking between following heuristics with "$*$" is significant and round brackets indicate results that are only roughly comparable.

pure random sampling using SSGS for schedule generation. Recently, a self-adaptive genetic algorithm was proposed in [15], where the proportion of individuals that construct schedules by applying the SSGS or the PSGS changes during the run of the algorithm. In this algorithm, local search is done when the best solution found by the GA has improved for several generations.

In [15] and [16], all heuristics were allowed to generate and evaluate at most 5000 schedules for each problem instance. The results mentioned in these studies are compared in Table II with AS-RCPSP. The table also contains results of the following two studies, which are based on computations that did not generate exactly 5000 schedules for each instance, but should roughly be comparable with respect to computational effort. The approach studied in [31] is based on a list scheduling heuristic that uses solutions obtained by Lagrangian relaxation of a time-indexed integer programming formulation. With this approach, for the

600 problem instances in j120.sm, a deviation from the lower bound of 36.2% was obtained when on the average 3675 solutions were generated per instance using an average time of 65 seconds per instance on a Sun Ultra 2 with a 200-MHz processor. A time-oriented branch-and-bound approach of [12] obtained for j120.sm an average deviation from the lower bound of 37.5% (37.1%) when using at most 60 s (respectively, 300 s) per instance on an Pentium Pro/200 PC. A "$*$" in the table indicates that the corresponding heuristic is significantly better than the next heuristic below it that is marked with a "$*$" (the test for significance was the Wilcoxon signed-rank test at level of significance of 5%; the tests for heuristics other than AS-RCPSP were done in [15]). Note that a lack of a "$*$" for the results of a heuristic means only that no significance test was done, i.e., it does not necessarily mean that there is no significant difference to the next heuristic below it.

Table II shows that AS-RCPSP with an average deviation of 35.43% performed better than all other heuristics. This average was obtained over ten runs for each of the 600 instances. The standard deviation is 0.046, the minimum value is 35.35%, and the maximum value of 35.50% is still better than the second best algorithm in table [15] (which achieved a deviation of 35.60%).

The results of two other heuristics for the RCPSP that were recently proposed in the literature are not included in Table II since the results for j120.sm are not directly comparable with respect to computational effort. The approach of [38] uses concepts of tabu search together with probabilistic fan search that is applied alternately to improve the head and the tail of a solution. Using a maximum computation time of 44 s per instance of j120.sm on a PC with AMD 400–MHz processor without restriction on the number of evaluated schedules, an average deviation from the lower bound of 34.53% was obtained. Another tabu search algorithm for the RCPSP was proposed in [32]. This algorithm found in 30 000 iteration steps for each of the instances in j120.sm solutions with an average deviation from the lower bound of 34.99% [38]. For 219 problems in j120.sm, a solution was found that was at least as good as the best-known solution at that time (June 1999) and for 50 instances, a new best solutions was found. One run of the algorithm took about 10 min on a Sun Ultra 2 (300 MHz, 1-GB memory). Compared to this, AS-RCPSP behaves very well. It found 186 best solutions (but compared to the improved bounds from January 2001) in about 25 s for one run on a Pentium III 500-MHz processor. It must be mentioned that AS-RCPSP was not designed specifically to obtain a fast processing time, but instead to give a good optimization behavior with respect to the number of generated solutions. With respect to running time, it might be advantageous to use methods that allow obtaining the new schedules very quickly by reusing parts of older schedules.

In the following, we study in more detail the various features of AS-RCPSP. The standard AS-RCPSP begins with a decision phase where one colony of forward ants and one colony of backward ants work for 100 generations (see Section VI-E). Using only forward ants results in an average deviation from the lower bounds of 35.94%, while using only backward ants (which is equivalent to using instances with reverse precedences) gives an average deviation of 35.77%. Since both results are significantly worse than the deviation of 35.43% for the standard AS-RCPSP, this shows that the algorithm can successfully decide after 100 generations which instance seems more profitable. On the average, the forward colonies in AS-RCPSP found in 38.6% of the cases the better result in the first 100 generations (with respect to the criterion described in Section VII). In 51.6% of the cases, the results of the backward ants were better and in the remaining 9.9% forward and backward ants were equally good. In general, the set of reversed instances from j120.sm seem to be slightly easier for the ant algorithm. This can be seen also in several figures shown in the rest of this section, but we do not discuss this fact again in the following. It should be mentioned that, in accordance to our speculation in Section VI-E, tests have shown that the criterion we proposed to decide which of the two colonies is allowed to proceed performed better than simply proceeding with the colony that has the best best-found solution.

Another feature of AS-RCPSP is the changing value of β that controls the influence of the heuristic (see Section VI-A). In AS-RCPSP, the value of β decreases linearly from 2 in the first generation to zero after 50% of all generations. Fig. 7 shows the behavior of AS-RCPSP when the value of β remains constant over the generations. The best average result with a deviation of 35.71% from the lower bounds was obtained for constant $\beta = 1.25$. This is significantly worse than the average deviation of 35.43% obtained with our strategy to change the β value.

Another parameter of AS-RCPSP that changes during a run is ρ (see Section VI-A). The value of $\rho = 0.025$ remains constant during the first generations and then switched to a larger value of $\rho = 0.075$ for the last 200 generations. Fig. 8 shows the behavior of AS-RCPSP when the value of ρ remains constant over all generations. The best average result with a deviation of 35.58% from the lower bounds was obtained for constant $\rho = 0.04$. This is significantly worse than the average deviation of 35.43% obtained with the changing ρ value.

AS-RCPSP uses a mechanism to restrict the influence of an elitist solution. Each elitist solution is discarded after $g_{\max} = 10$ generations in order to give an elitist solution not too much influence (see Section VI-C). Fig. 9 shows the behavior of AS-RCPSP for different values of $g_{\max}$. The results show that $g_{\max}$ has a clear effect on the optimization behavior. A high value of $g_{\max}$ leads to an average deviation from the lower bounds of more than 36.0%, but also too small values are not good, e.g., for $g_{\max} = 3$ gives an average deviation of 35.77%.

The influence of the final local optimization steps in AS-RCPSP is depicted in Fig. 10. The figure shows that the local optimization steps have a small positive influence on the optimization behavior. Without local optimization, AS-RCPSP obtained an average deviation from the lower bounds of 35.51% compared to a deviation of 35.43% for 250 local optimization steps. Since only 5000 schedule evaluations were allowed in each run of our tests a growing number of local steps reduces the allowed number of iterations of the ants. Hence, too much local optimization steps lead to a worse behavior, e.g., for 1000 local optimization steps the average deviation was 35.56%.

We also computed results for AS-RCPSP when the local optimization heuristic of [15] was used instead of our standard heuristic that swaps two activities (see Section VI-D). With 250

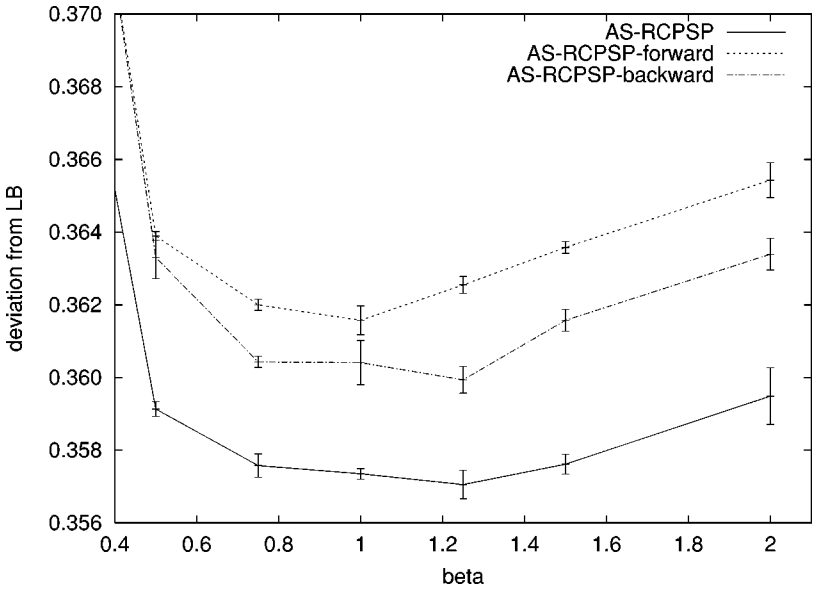

Fig. 7. AS-RCPSP with constant values of β (bars indicate the standard deviation).

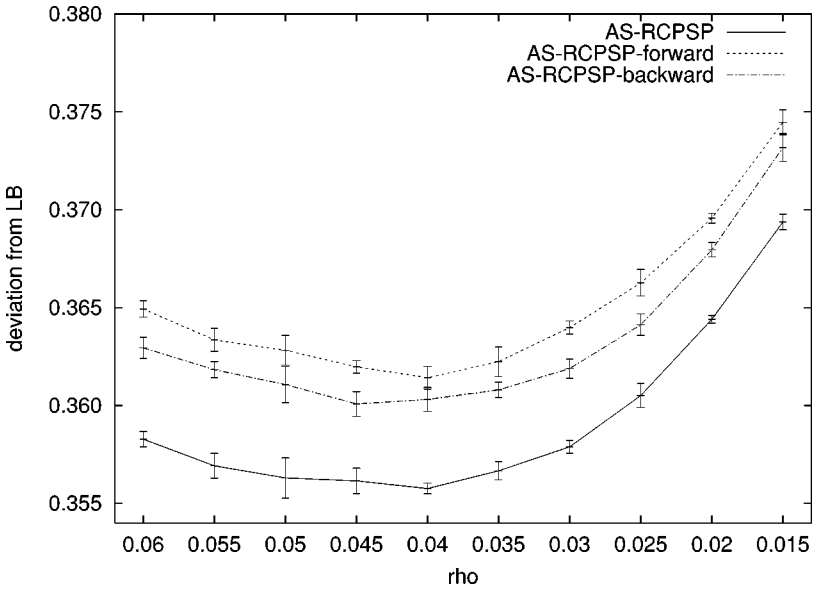

Fig. 8. AS-RCPSP with constant values of ρ (bars indicate the standard deviation).

steps of the heuristics from [15] at the end of a run, the average deviation over four runs of AS-RCPSP from the lower bounds was 35.46% with a standard deviation of 0.088. Thus, the results when using our heuristics are only slightly better.

The results of AS-RCPSP for different deterministic heuristics are shown in Table III. With an average deviation of 35.64%, the nLFT heuristic performs slightly worse than the similar nLST heuristic that is used in the standard AS-RCPSP algorithm. Also, nGRPWA performs well with an average deviation of 35.79%. Heuristic nMTS gives only moderate performance while the nWRUP is worse for all values of parameter ($\omega \in \{0, 0.5, 1\}$). All these results fit well to the results in [19], where a similar ranking between the heuristics according to their optimization behavior in connection with the SSGS was obtained.

To further investigate the potential of our ACO algorithm, we tested s-AS-RCPSP with additional local search steps after every generation of ants on the benchmark set j120.sm. The parameters used for these tests were $m = 20$ ants, $\rho = 0.005$, a maximum of 20 000 iterations, and an application of the local search on the best solution in every generation. For these values, we found an average deviation from the critical path lower bounds of 32.97%. For 278 of the 600 test instances, a solution was found that is at least as good as the known best solution.

A further proof of the strength of the algorithm is that it was able to find new best solutions for 130 problem instances of the j120.sm problem set when no restriction on the number of evaluated schedules was set. These are nearly 1/3 of all problem instances that were not known to be solved optimally before.

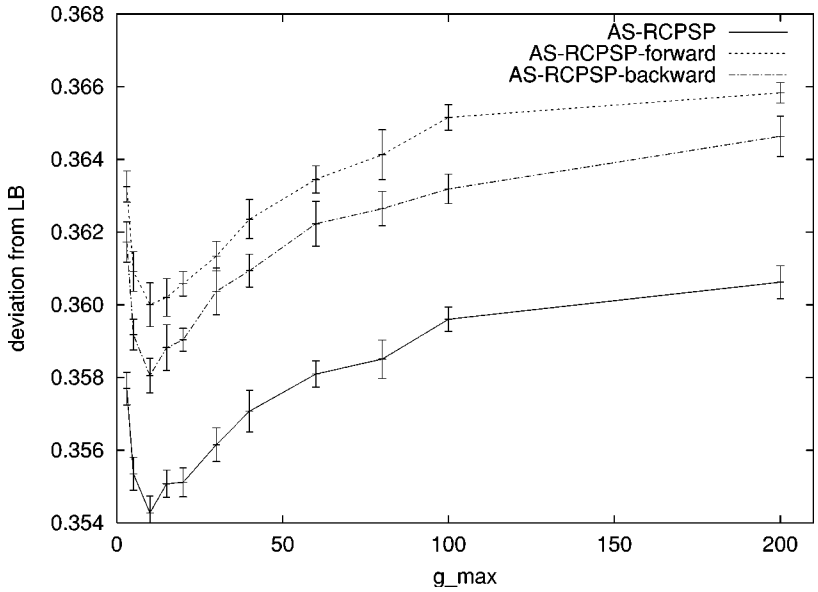

Fig. 9. AS-RCPSP with different values of $g_{\max}$ (bars indicate the standard deviation).

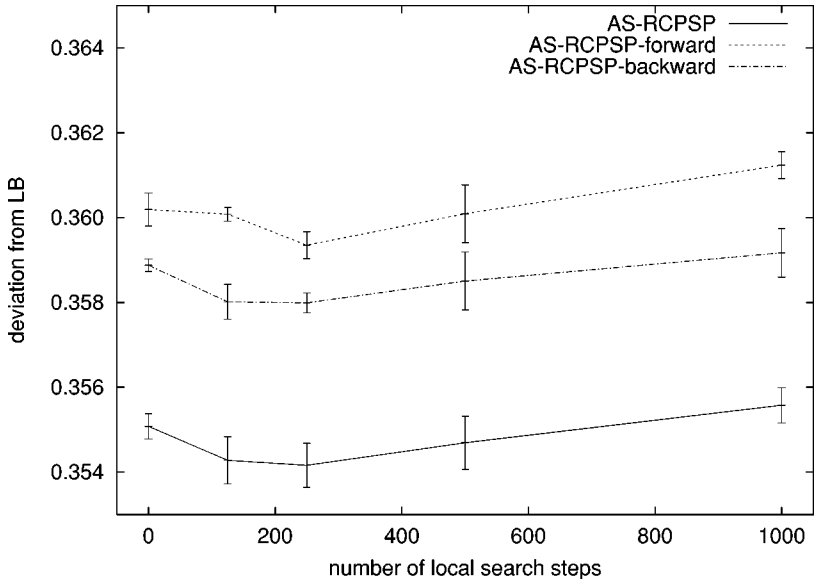

Fig. 10. AS-RCPSP with different number of maximal final local optimization steps (bars indicate the standard deviation).

Finally, it should be noted that some parameters of our algorithm are only of importance when the number of generations is restricted. For example, instead of using two colonies (one with forward ants and one with backward ants) for the first generations, it is possible to run the algorithm twice when there is enough time (once with forward ants and once with backward ants). The parameter ρ can be set constant instead of switching it to a greater value after several generations. In general, the smaller ρ is, the longer it will take the algorithm to converge and the better will be the quality of the obtained solution. The number of local search steps at the end should not be too small (after a few hundred steps the probability of finding better solutions with more steps becomes quite small). The larger the number of ants per generation, the better will usually be the best-found solution, but a few dozen of ants per generation should be enough in most cases. Summation evaluation should have a signficant influence ($c = 0.5$ is a good value) and the elitist ant should be forgotten after every small number of generations. During the first generations, the heuristic should have at least the same influence as the pheromone (e.g., $\alpha = 1$, $\beta \geq 1$), but in later stages of a run, the influence of the heuristic should be much less than the influence of the pheromone. It has to be investigated whether forgetting the elitist ant after every few generations and shrinking the influence of β during a run is good for ACO in general or only for specific problems like the RCPSP.

IX. Conclusion

We have introduced an ACO approach for RCPSP. The approach has several new features that are intersting for ACO in general. A combination of direct (or local) and summation (or

TABLE III
AS-RCPSP WITH DIFFERENT HEURISTICS

Heuristic	deviation from LB in %	standard deviation
nLST	35.43	0.046
nLFT	35.64	0.043
nGRPWA	35.79	0.049
nMTS	36.38	0.004
nWRUP ($\omega = 0$)	39.22	0.080
nWRUP ($\omega = 0.5$)	38.32	0.082
nWRUP ($\omega = 1$)	38.70	0.037

Average deviation from lower bound.

global) pheromone evaluation methods is used by the ants for the construction of a new solution. Additional features of the algorithm are the changing strength of heuristic influence, the changing rate of pheromone evaporation over the ant generations, and the restricted influence of the elitist solution by forgetting it at regular intervals. The value of all new features has been shown through extensive experiments. We compared our approach with the results of various other randomized heuristics for the RCPSP including genetic algorithms and simulated annealing on the set of largest instances in the benchmark library PSPLIB. Under the constraint that every algorithm is allowed to compute and evaluate the same restricted number of solutions, our algorithm performed best. Moreover, allowing more evaluations, our algorithm was able to find 130 new best solutions. That means that nearly 1/3 of the 396 instances in the benchmark set that were not known to be solved optimally have been improved. The fact that the algorithm behaves very well (compared to several other heuristics) in both cases—with and without restrictions to the number of evaluated schedules—shows the flexibility of the approach.

ACKNOWLEDGMENT

The authors would like to thank M. Dorigo, D. B. Fogel, and the referees for their comments.

REFERENCES

[1] A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss, "An ant colony optimization approach for the single machine total tardiness problem," in *Proc. 1999 Congr. Evolutionary Computation*, 1999, pp. 1445–1450.
[2] H. M. Bottee and E. Bonabeau, "Evolving ant colony optimization," *Adv. Complex Syst.*, vol. 1, no. 2/3, pp. 149–159, 1998.
[3] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource constrained project scheduling problem," Service de Robotique et Automatization, Univ. de Liège, Liège, Belgium, 1998.
[4] P. Brucker, A. Drexel, R. H. Möhring, K. Neumann, and E. Pesch, "Resource-constraint project scheduling: Notation, classification, models, and methods," *Eur. J. Oper. Res.*, vol. 112, no. 1, pp. 3–41, Jan. 1999.
[5] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian, "Ant system for job-shop scheduling," *Belg. J. Oper. Res. Stat. Comput. Sci.*, vol. 34, no. 1, pp. 39–53, 1994.
[6] E. W. Davis and J. H. Patterson, "A comparison of heuristic and optimum solutions in resource-constrained project scheduling," *Manage. Sci.*, vol. 21, no. 8, pp. 944–955, Apr. 1975.
[7] M. L. den Besten, T. Stützle, and M. Dorigo, *Ant Colony Optimization for the Total Weighted Tardiness Problem*. Berlin, Germany: Springer-Verlag, 2000, vol. 1917, Lecture Notes in Computer Science, pp. 611–620.
[8] M. Dorigo, "Optimization, learning, and natural algorithms," Ph.D. dissertation (in Italian), Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.
[9] M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. New York: McGraw-Hill, 1999, pp. 11–32.
[10] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 53–66, Apr. 1997.
[11] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst. Man Cybern. B*, vol. 26, pp. 29–41, Feb. 1996.
[12] U. Dorndorf, E. Pesch, and T. Phan Huy, "A branch and bound algorithm for the resource-constrained project scheduling problem," *Math. Methods Oper. Res.*, vol. 51, no. 3, pp. 565–594, 2000.
[13] S. E. Elmaghraby, *Activity Networks*. New York: Wiley, 1977.
[14] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Nav. Res. Logist.*, vol. 45, no. 7, pp. 733–750, 1998.
[15] ——, "Self-adapting genetic algorithm with an application to project scheduling," Univ. of Kiel, Kiel, Germany, Tech. Rep. 506, 1999.
[16] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 127, no. 2, pp. 394–407, Dec. 2000.
[17] W. Herroelen, B. De Reyck, and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments," *Comput. Oper. Res.*, vol. 13, no. 4, pp. 279–302, 1998.
[18] S. Iredi, D. Merkle, and M. Middendorf, "Bi-criterion optimization with multi colony ant algorithms," in *First International Conference on Evolutionary Multi-Criterion Optimization (EMO'01)*. Berlin, Germany: Springer-Verlag, 2001, vol. 1993, Lecture Notes in Computer Science, pp. 359–372.
[19] R. Klein, "Bidirectional planning: Improving priority rule-based heuristics for scheduling resource-constrained projects," *Eur. J. Oper. Res.*, vol. 127, no. 3, pp. 619–638, Dec. 2000.
[20] R. Kolisch, "Production and logistics," in *Project Scheduling Under Resource Constraints*. Heidelberg, Germany: Physica-Verlag, 1995.
[21] ——, "Effcient priority rules for the resource-constrained project scheduling problem," *J. Oper. Manage.*, vol. 14, no. 3, pp. 179–192, 1996.
[22] ——, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation," *Eur. J. Oper. Res.*, vol. 90, no. 2, pp. 320–333, 1996.
[23] R. Kolisch and A. Drexl, "Adaptive search for solving hard project scheduling problems," *Naval Res. Logist.*, vol. 43, no. 1, pp. 23–40, 1996.
[24] R. Kolisch and S. Hartmann, "Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis," in *Handbook on Recent Advances in Project Scheduling*, J. Weglarz, Ed. Amsterdam, The Netherlands: Kluwer, 1999, pp. 197–212.
[25] R. Kolisch, C. Schwindt, and A. Sprecher, "Benchmark instances for project scheduling problems," in *Handbook on Recent Advances in Project Scheduling*, J. Weglarz, Ed. Amsterdam, The Netherlands: Kluwer, 1999, pp. 147–178.
[26] R. Kolisch and A. Sprecher, "PSPLIB – A project scheduling problem library," *Eur. J. Oper. Res.*, vol. 96, no. 1, pp. 205–216, 1996.
[27] V. J. Leon and B. Ramamoorthy, "Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling," *OR Spektrum*, vol. 17, no. 2/3, pp. 173–182, 1995.
[28] D. Merkle and M. Middendorf, "An ant algorithm with a new pheromone evaluation rule for total tardiness problems," in *Proceedings of the EvoWorkshops 2000*. Berlin, Germany: Springer-Verlag, 2000, vol. 1803, Lecture Notes in Computer Science, pp. 287–296.
[29] ——, "A new approach to solve permutation scheduling problems with ant colony optimization," in *Second European Workshop on Scheduling and Timetabling (EvoStim2001)*. Berlin, Germany: Springer-Verlag, 2001, vol. 2037, Lecture Notes in Computer Science, pp. 484–493.
[30] 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. New York: McGraw-Hill, 1999, pp. 51–61.
[31] R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz, "Solving Project Scheduling Problems by Minimum Cut Computations," Dept. Mathematics, Tech. Univ. Berlin, Berlin, Germany, Tech. Rep. 680-2000, 2000.

[32] K. Nonobe and T. Ibaraki, "Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP)," Dept. Applied Mathematics and Physics, Kyoto Univ., Kyoto, Japan, 1999.
[33] A. Schirmer, "Case-based reasoning and improved adaptive search for project scheduling," *Naval Res. Logist.*, vol. 47, no. 3, pp. 201–222, 2000.
[34] J. P. Stinson, E. W. Davis, and B. M. Khumawala, "Multiple resource-constrained scheduling using branch and bound," *AIIE Trans.*, vol. 10, pp. 252–259, 1978.
[35] T. Stützle, "An ant approach for the flow shop problem," in *Proc. 6th Eur. Congr. Intelligent Techniques and Soft Computing*, vol. 3, Aachen, Germany, 1998, pp. 1560–1564.
[36] T. Stützle and H. H. Hoos, "*Max-min* ant system," *Fut. Gener. Comput. Syst.*, vol. 16, no. 8, pp. 889–914, June 2000.
[37] G. Ulusoy and L. Özdamar, "Heuristic performance and network/resource characteristics in resource-constrained project scheduling," *J. Oper. Res. Soc.*, vol. 40, pp. 1145–1152, 1989.
[38] V. Valls, S. Quintanilla, and F. Ballestin, "Resource-constrained project scheduling: A critical activity reordering heuristic," unpublished.
[39] S. van der Zwaan and C. Marques, "Ant colony optimization for job shop scheduling," presented at the Third Workshop on Genetic Algorithms and Artificial Life, Lisbon, Portugal, 1999.

Daniel Merkle received the Dipl. degree in computer science from the University of Karlsruhe, Karlsruhe, Germany, in 1997. He is currently working toward the Ph.D. degree in applied computer science at the same university.

He is a Research Assistant with the Institute of Applied Informatics and Formal Description Methods, University of Karlsruhe. His current research interests include ant colony optimization, multicriterion optimization, reconfigurable architectures, and cellular automata.

Martin Middendorf received the Dipl. degree in mathematics and the Dr. rer. nat. from the University of Hannover, Hannover, Germany, in 1988 and 1992, respectively, and the Habilitation from the University of Karlsruhe, Karlsruhe, Germany, in 1998.

He was a Visiting Professor of Computer Science with the University of Dortmund, Dortmund, Germany, and the University of Hannover. He is currently a Professor of Computer Science with the Catholic University of Eichstätt-Ingolstadt, Eichstätt, Germany. His current research interests include algorithms from nature, parallel algorithms, scheduling, and reconfigurable architectures.

Hartmut Schmeck received the Dipl. degree, Dr. rer. nat., and Habilitation in computer science from the University of Kiel, Kiel, Germany, in 1975, 1981, and 1990, respectively.

Since 1991, he has been a Professor of Applied Computer Science with the University of Karlsruhe, Karlsruhe, Germany. He has also held visiting positions at Queen's University, Canada, Hildesheim and Münster, Germany, Copenhagen, Denmark, and Newcastle, Australia. His current research interests include the efficient use of computing resources for solving computationally expensive problems, particularly evolutionary computation and the design and analysis of parallel and distributed algorithms and architectures. Recently, he started several projects on teleteaching and e-learning.