Ant Colony Optimization for FOP Shop scheduling: A case study on different pheromone representations

Christian Blum and Michael Sampels
Université Libre de Bruxelles, IRIDIA
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium
{cblum,msampels}@ulb.ac.be

Abstract - In this work we deal with the FOP Shop scheduling problem which is a general scheduling problem including Job Shop scheduling, Open Shop scheduling and Mixed Shop scheduling as special cases. The aim of this paper is to compare different pheromone representations taken from the literature with our new approach. The pheromone representations are used by an Ant Colony Optimization algorithm to construct solutions to the FOP Shop scheduling problem.

I. FOP Shop scheduling

Ant Colony Optimization (ACO) [5] is a recently proposed meta-heuristic approach for solving hard combinatorial optimization problems. The inspiring source of ACO is the foraging behavior of real ants. In this work we compare different pheromone representations for ACO to tackle instances of the general class of FOP Shop scheduling problems. The class of FOP Shop scheduling problems (**F**irst **O**rder **P**arallel Shop scheduling) – in the following called FOPSSP – was formulated in the course of the Metaheuristics Network [1] [1] as a generalization of scheduling problems covering the well known Job Shop scheduling problem (JSSP), the Open Shop scheduling problem (OSSP) and the Mixed Shop Scheduling problem (MSSP) [9], [3]. An instance of the the FOP Shop scheduling problem consists of the following elements:

- A set $\mathcal{O} = \{o_1, ..., o_n\}$ of *operations* which is partitioned into *jobs* $\mathcal{J}_1, \dots, \mathcal{J}_r$. For $o \in \mathcal{O}$, we denote $o \in \mathcal{J}_i$ by $j(o) = i$.
- The partition of $\mathcal{O}$ into jobs is further refined to a partition into *groups* $\mathcal{G}_1, \dots, \mathcal{G}_g$. For a group $\mathcal{G}$, we denote $\mathcal{G} \subset \mathcal{J}_i$ by $j(\mathcal{G}) = i$.
- $\mathcal{O}$ is partitioned into *machines* $\mathcal{M}_1, \dots, \mathcal{M}_m$. For $o \in \mathcal{O}$, we denote $o \in \mathcal{M}_i$ by $m(o) = i$. $m(o)$ represents the machine on which an operation o has to be processed. Its processing time is given by $pt(o) \in \mathbf{N}$.
- The groups of each job are totally ordered. If in such an order a group $\mathcal{G}_i$ is placed before a group $\mathcal{G}_k$, we write $\mathcal{G}_i \prec \mathcal{G}_k$. The immediate predecessor of a group $\mathcal{G}_i$ is denoted by $\prec_{pred} (\mathcal{G}_i)$. The immediate successor of a group $\mathcal{G}_i$ is denoted by $\prec_{succ} (\mathcal{G}_i)$. The order of groups within jobs induces precedence constraints among the operations of jobs. For a pair of operations o_i, o_k with $j(o_i) = j(o_k)$ holds: o_i has to be processed before o_k iff $g(o_i) \prec g(o_k)$.

The JSSP is a special case of the FOPSSP in the sense that an instance of the JSSP can be considered as an instance of the FOPSSP where each group contains only one operation. The FOPSSP also covers all OSSP instances in the sense that an OSSP instance can be treated as a FOPSSP instance where the groups are identical with the jobs. In Mixed Shop scheduling the set of jobs decomposes into two disjoint subsets. In one subset the jobs are handled as the jobs in Open Shop scheduling, whereas in the other subset the jobs are handled as in Job Shop scheduling. Therefore the FOPSSP also contains the MSSP as a special case. As the JSSP, the OSSP, and the MSSP are in general NP-hard problems, we can also conclude that the FOPSSP is in general NP-hard.

An instance to any scheduling problem can be visualized by the vertex weighted disjunctive graph representation for scheduling problems introduced by Roy and Sussmann [11].

Definition 1: A *vertex weighted disjunctive graph* G is a three-tuple (V, A, E) where V is a set of vertices, A a set of directed arcs, and E a set of undirected edges. A function f is given which assigns a weight $f(v) \in \mathbf{N}$ to every $v \in V$.

For the FOPSSP the disjunctive graph $G = (V, A, E)$ is defined as follows. Every vertex $v \in V$ corresponds to an operation $o \in \mathcal{O}$, and $f(v)$ is defined as $pt(o)$ of

[1]The Metaheuristics Network is a Research Training Network funded by the Improving Human Potential program of the CEC. The Network aims at the comparison of metheuristics on different combinatorial optimization problems.

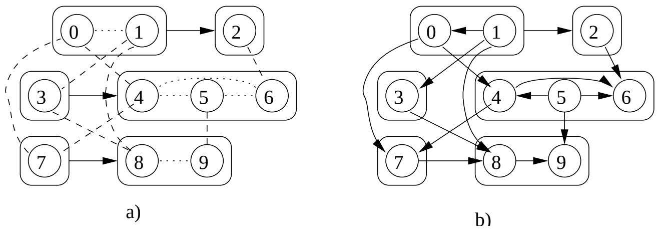

Fig. 1. a) shows the disjunctive graph representation of a simple instance of the FOPSSP with 3 jobs, 6 groups and 10 operations (processing times are omitted in this example). The dashed and dotted links are the elements of set E. The directed arcs between operations of successive groups (set A) are simplified as inter-group connections. b) shows a solution to the problem (the arcs of set E are directed such that the resulting graph doesn't contain any cycles).

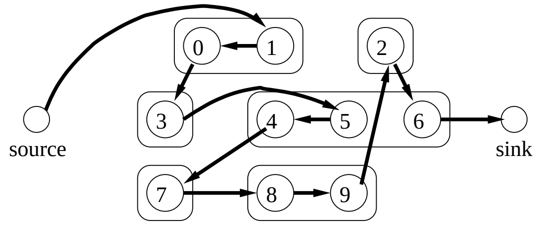

Fig. 2. An ant walk from source to sink for the example presented in Fig. 1. This ant walk orders the operations as follows: 1-0-3-5-4-7-8-9-2-6. This order implicitly defines orders in groups (1-0, 5-4-6, 8-9) and on machines (0-4-7, 1-3-8, 2-6, 5-0). This means that the ant walk in this figure defines the direction of the arcs of the disjunctive graph representation as shown in Fig. 1b)

the corresponding operation (in the following we denote the vertices by their operations). A, the set of directed arcs, is defined as $A = \{(o_i, o_j) \mid o_i, o_j \in \mathcal{O},\, g(o_i) = \prec_{pred}(g(o_j))\}$, and E is the set of undirected arcs defined by $E = \{(o_i, o_j) \mid o_i, o_j \in \mathcal{O},\, g(o_i) = g(o_j) \vee m(o_i) = m(o_j)\}$.

Every solution to an instance of the FOPSSP can be represented by a version of this graph where every arc in E has been given a direction such that the resulting directed graph is acyclic. By giving directions to the arcs in E we specify processing orders for the operations in groups and on machines. The value (or quality) of a solution is equivalent to the length of the longest path in G (where the length of the path is defined as the sum of the processing times of the operations on this path). This value is often called the *minimum makespan.* See Fig. 1 for an example.

II. The ACO algorithm for the FOPSSP

In this section we outline the framework of our ACO algorithm for the FOPSSP. Our aim was not to produce a highly sophisticated algorithm with a lot of parameters to tune. We rather wanted to define a simple ACO algorithm as a framework for the comparison of different pheromone representations. The basic framework of our algorithm is shown in Alg. 1. Below all the components of this algorithm are being explained. In algorithm 1, $\tau = \{\tau_1, ..., \tau_l\}$ is a set of pheromone values, k is the number of ants, and s^j are solutions to the problem (ordered lists of all the operations), constructed by the ants.

InitializePheromoneValues(τ): In every version of our algorithm we initialize all the pheromone values to the same positive constant value.

ConstructSolution(τ): To tackle the FOPSSP with an ACO algorithm we have to define the constructive heuristic to be used in a probabilistic manner to construct solutions to the problem. In ACO algorithms artificial ants construct a solution by building a path on a *construction graph* $\mathcal{G}_c = (\mathcal{C}, \mathcal{L})$ where the elements of the set $\mathcal{C}$ (called *components*) and the elements of the set $\mathcal{L}$ (called *links*) are given for the Fop Shop scheduling problem as follows:

$$
\begin{aligned}
\mathcal{C} &= \mathcal{O} \cup \{v_{source}, v_{sink}\} \\
\mathcal{L} &= \{(o_i, o_j) \mid o_i, o_j \in \mathcal{O}\} \cup \{(v_{source}, o_i) \mid o_i \in \mathcal{O}\} \\
&\cup \quad \{(o_i, v_{sink}) \mid o_i \in \mathcal{O}\}
\end{aligned}
$$

Note that all links in $\mathcal{L}$ are directed. This graph is basically the fully connected graph of operations plus a source node (and arcs from the source node to every operation) and a sink node (and arcs from every operation to the sink node).

To build a solution an ant starts in the source node and traverses the graph $\mathcal{G}_c$ visiting every node exactly once, finishing in the sink node. This corresponds to building a permutation (or an ordered list) of all the operations which uniquely defines the processing orders of the operations in groups and on machines (see Fig. 2). Not every possible path of the kind mentioned above corresponds to a feasible solution (some paths might result in solutions where precedence constraints are

Algorithm 1 ACO for the FOPSSP

```
InitializePheromoneValues(τ)
while termination conditions not met do
  for j = 1 to k do
    s^j ← ConstructSolution(τ)
  end for
  ApplyOnlineDelayedPheromoneUpdate(τ,s^1,...,s^k)
end while
```

violated, which corresponds to having cycles in the directed version of the disjunctive graph representation). To avoid generating unfeasible solutions we use a *list scheduler* algorithm to restrict the set of operations the ant is allowed to walk to in the next step. A well known member of the class of list scheduler algorithms is the heuristic proposed by Giffler and Thompson [6]. List scheduler algorithms generate an ordered list of all the operations of a problem instance in a step-by-step manner from left to right. In every step a subset of the yet unscheduled operations is generated, which when scheduled at that point keep the partial schedule feasible. Then, to some criteria, one of these operations is chosen and appended at the end of the list of already scheduled operations. List scheduler algorithms (depending on the criteria mentioned above) produce active or non-delay schedules. Algorithm 2 shows the ant construction phase using the above mentioned mechanism of a list scheduler algorithm to restrict the set of operations where the ant is allowed to go in the next step. It remains to be specified how the probabilities to choose the next operation from the set of operations provided by the list scheduler mechanism are generated. This is dependent on the pheromone representations which are outlined in the following section.

Algorithm 2 Ant construction phase

```
t = 1
Ant is placed on v_source of graph G_c
s = empty list with n positions (n being the number of operations)
J_t = {o_i | g(o_i) doesn't have a predecessor}
for t = 1 to n do
   Choose o*_i ∈ J_t to probability p(o*_i|s,t)
   Ant moves to operation o*_i
   s[t] = o*_i
   if {o_i ∈ J_t | g(o_i) = g(o*_i)} = ∅ AND ≺_succ (g(o*_i)) exists then
      J_{t+1} = J_t \ {o*_i} ∪ {o_j | o_j ∈ ≺_succ (g(o*_i))}
   else
      J_{t+1} = J_t \ {o*_i}
   end if
end for
Ant moves to v_sink of graph G_c
```

ApplyOnlineDelayedPheromoneUpdate(τ,s^1,...,s^k): We implemented our ACO algorithm in the hyper-cube framework introduced by Blum et al. [2]. For a set of pheromone values $\{\tau_1, ..., \tau_n\}$ in ACO algorithms we have a pheromone updating rule looking usually like that:

$$\tau_i \leftarrow (1-\rho) \cdot \tau_i + \sum_{j=1}^{k} \Delta^{s^j} \tau_i \tag{1}$$

where

$$\Delta^{s^j} \tau_i = \begin{cases} f(s^j) & \text{if } s^j \text{ contributes to } \tau_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\Delta^{s^j}\tau_i$ is the contribution of a solution s^j to the update for pheromone value τ_i (k is the number of solutions used for updating the pheromones), ρ is the evaporation rate (a small positive constant), and f is function (it usually maps the quality of a solution to its inverse). In the hyper-cube framework a normalization of the contribution of every solution used for updating the pheromone values is done in the following way:

$$\tau_i \leftarrow (1-\rho) \cdot \tau_i + \rho \cdot \sum_{j=1}^{k} \Delta^{s^j} \tau_i \tag{3}$$

where

$$\Delta^{s^j} \tau_i = \begin{cases} \frac{f(s^j)}{\sum_{l=1}^{k} f(s^l)} & \text{if } s^j \text{ contributes to } \tau_i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where we multiply the normalized sum of contributions with the evaporation rate ρ. This formula can be reformulated as:

$$\tau_i \leftarrow \tau_i + \frac{\rho}{\sum_{l=1}^{k} f(s^l)} \left(\sum_{j=1}^{k} f(s^j) \cdot \delta(s^j, \tau_i) - \tau_i \right) \tag{5}$$

where

$$\delta(s^j, \tau_i) = \begin{cases} 1 & \text{if } s^j \text{ contributes to } \tau_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

This leads to a scaling of the objective function values and the pheromone values are implicitly limited to the interval $[0, ..., 1]$ (see [2] for a more detailed description). The exact updating rules are dependent on the different pheromone representations and are outlined in Sec. IV.

III. Different pheromone representations

There are a number of design decisions to be made when developing an ACO algorithm to tackle a combinatorial optimization problem. One of the most crucial choices is the modeling of the set of pheromones. For the TSP for example it is a fairly obvious choice to put a pheromone value on every link between a pair of cities. For other combinatorial optimization problems the choice is not as obvious as for the TSP (see [10] for an example of different models for MAX-SAT). Often this problem can be stated as the problem of assigning pheromone values to the decision variables themselves (first order pheromone values) or to subsets of decision variables (higher order pheromone values). In the following we present four different pheromone representations, three of them from the literature, and a new one.

Learning of absolute positions in s: This first pheromone model (henceforth$\mathsf{PH}_{\mathsf{abs}}$) is the most simple choice of a pheromone representation and can be regarded as a standard in permutation type problems. To every operation $o_j \in \mathcal{O}$ and every position i in list s we have associated a pheromone value $\tau_{o_j,i}$. The probabilities $p(o_j|s,t)$ for the operations in set J_t of the ant construction phase to be chosen by the ant (called transition probabilities) are determined as follows:

$$\begin{cases} \frac{\tau_{o_j,t}}{\sum_{o_k \in J_t} \tau_{o_k,t}} & : \text{ if } \quad o_j \in J_t \\ 0 & : \text{ otherwise} \end{cases}$$

where s is the ordered list of operations scheduled so far (the partial schedule constructed so far), and J_t is the set of operations allowed to be scheduled next (see Alg. 2). With this pheromone representation the algorithm tries to learn absolute positions of operations in s.

Learning of absolute positions in s plus summing evaluation: The pheromone model in this case (henceforth $\mathsf{PH}_{\mathsf{sum}}$) is the same as in learning absolute positions, except that the evaluation of the transition probabilities $p(o_j|s,t)$ is different. It was introduced in [7] and further tested in [8]. The transition probabilities $p(o_j|s,t)$ are:

$$\begin{cases} \frac{\sum_{l=1}^{t} \tau_{o_j,l}}{\sum_{o_k \in J_t} \sum_{l=1}^{t} \tau_{o_k,l}} & : \text{ if } \quad o_j \in J_t \\ 0 & : \text{ otherwise} \end{cases}$$

where s and J_t are as described above. In this way of pheromone evaluation, if an operation is by some stochastical error not placed at a position in s where it should have been placed, the probability remains high to schedule it closely afterward.

Learning of a predecessor relation in s: This pheromone model (henceforth $\mathsf{PH}_{\mathsf{suc}}$) was introduced by Colorni et al. [4] for an Ant System to tackle the Job Shop scheduling problem. In the following the simple extension of this model to the Fop Shop scheduling problem is outlined. In this model we have a pheromone value τ_{o_i,o_j} on every pair of operations $o_i, o_j \in \mathcal{O}$ and we have pheromone values τ_{v_{source},o_i} $\forall o_i \in \mathcal{O}$. The transition probabilities $p(o_j|s,t)$ are determined as follows:

$$\begin{cases} \frac{\tau_{v_{source},o_j}}{\sum_{o_k \in J_t} \tau_{v_{source},o_k}} & : \text{ if } o_j \in J_t, t = 1 \\ \frac{\tau_{o_i,o_j}}{\sum_{o_k \in J_t} \tau_{o_i,o_k}} & : \text{ if } o_j \in J_t, t > 1, s[t-1] = o_i \\ 0 & : \text{ otherwise} \end{cases}$$

where s and J_t are as described above. In this way of modeling the pheromones in every step of the construction phase the next operation to be scheduled is dependent on the operations scheduled in the previous step.

Learning of relations among operations: In this new pheromone model (henceforth $\mathsf{PH}_{\mathsf{rel}}$) we assign pheromone values to pairs of *related* operations. We call two operations $o_i, o_j \in \mathcal{O}$ *related* if they are in the same group, or if they have to be processed on the same machine. Formally, a pheromone value

$$\tau_{o_i,o_j} \;\text{ exists, iff }\; g(o_i) = g(o_j) \;\text{ or }\; m(o_i) = m(o_j)$$

The meaning of a pheromone value τ_{o_i,o_j} is, that if τ_{o_i,o_j} is high then operation o_i should be scheduled before operation o_j. The choice of the next operations to schedule is handled as follows. If there is an operation $o_i \in J_t$ with no related and unscheduled operations left, it is chosen. Otherwise we choose among the operations of set J_t with the following transition probabilities $p(o_j|s,t)$:

$$\begin{cases} \frac{\min_{o_r \in S_{o_j}^{rel}} \tau_{o_j,o_r}}{\sum_{o_k \in J_t} \min_{o_r \in S_{o_k}^{rel}} \tau_{o_k,o_r}} & : \text{ if } \quad o_j \in J_t \\ 0 & : \text{ otherwise} \end{cases}$$

where s and J_t are as described above, and $S_{o_j}^{rel} = \{o_k \in \mathcal{O} \mid m(o_k) = m(o_j) \vee g(o_k) = g(o_j),\, o_k \text{ not scheduled yet}\}$. The meaning of this rule to compute the transition probabilities is the following: If at least one of the pheromone values between an operation $o_i \in J_t$ and a related operations o_r (not scheduled yet) is low, this means that operation o_i probably shouldn't be scheduled now. By using this pheromone model the algorithm tries to learn relations between operations. The actual position of an operation in the ordered list s is not important anymore. It is the relative position of an operation with regard to the related operations.

IV. Pheromone update

As mentioned above, our ACO algorithm is implemented in the hyper-cube framework [2]. Therefore we get the following pheromone updating rules. For $\mathsf{PH}_{\mathsf{abs}}$ and $\mathsf{PH}_{\mathsf{sum}}$:

$$\tau_{o_i,j} \leftarrow \tau_{o_i,j} + \frac{\rho}{\sum_{l=1}^{k} f(s^l)} \left(\sum_{r=1}^{k} f(s^r) \cdot \delta(s^r, \tau_{o_i,j}) - \tau_{o_i,j} \right)$$

for $i, j = 1, ..., n$, the evaporations rate $\rho > 0$, and where $\delta(s^r, \tau_{o_i,j}) = 1$ if $s^r[j] = o_i$ and $\delta(s^r, \tau_{o_i,j}) = 0$ otherwise. For $\mathsf{PH_{suc}}$ the pheromone updating rule is as follows:

$$\tau_{o_i,o_j} \leftarrow$$
$$\tau_{o_i,o_j} + \frac{\rho}{\sum_{l=1}^{k} f(s^l)} \left(\sum_{r=1}^{k} f(s^r) \cdot \delta(s^r, \tau_{o_i,o_j}) - \tau_{o_i,o_j} \right)$$

for $i, j = 1, ..., n$ and where $\delta(s^r, \tau_{o_i,o_j}) = 1$ if $s[l] = o_i$ and $s[l+1] = o_j$ for one $l \in [1, ..., n-1]$ and $\delta(s^r, \tau_{o_i,o_j}) = 0$ otherwise. In the same way the pheromone update is done for all pheromones τ_{v_{source},o_j} for $j = 1, ..., n$ and where $\delta(v_{source}, o_j) = 1$ if $s[1] = o_j$ and $\delta(v_{source}, o_j) = 0$ otherwise.
For $\mathsf{PH_{rel}}$ the pheromone updating rule is as for $\mathsf{PH_{suc}}$, except that we only have pheromones for o_i and o_j related. In $\mathsf{PH_{rel}}$, $\delta(s^r, \tau_{o_i,o_j}) = 1$ if o_i is placed before o_j in s^r, $\delta(s^r, \tau_{o_i,o_j}) = 0$ otherwise. This way of generating the transition probabilities favors operations among those to be scheduled next for which the minimum of the pheromones to unscheduled related operations is maximal. In other words, if there is an operation among the candidates where this minimum is quite low, this means that there is at least one related operation which should be scheduled before.

V. Comparison

We were running experiments of our ACO algorithm for all four pheromone representations on several FOPSSP instances. The results are consistent over the whole set of problem instances tested. Due to space limitations we restrict ourselves to present only the results for the biggest problem tested. This problem instance is called whizzkids97.fss and consists of 197 operations. It was provided by the TU Eindhoven where it was subject of a competition held in 1997. For all 4 pheromone representations we were running experiments for a range of evaporation rates in order to find the best evaporation rate for every pheromone representation. A summary of the results of these experiments is shown in Fig. 3 (best values for ρ for all pheromone representations). Results for all tested values for ρ are shown in Fig. 4 separately for all pheromone representations. There are several observations worth to be mentioned. First of all, our new pheromone representation $\mathsf{PH_{rel}}$ shows a clear advantage over the other three pheromone representations. Further on, $\mathsf{PH_{abs}}$, $\mathsf{PH_{suc}}$ and $\mathsf{PH_{sum}}$ show (for most evaporation rates) a decrease in performance at least at the beginning of the search. A reason for this decrease in performance might be, that the problem model introduced by these pheromone representations makes good solutions very fragile, which drives the algorithm initially to bad solutions. For instance, we noticed that the algorithm using $\mathsf{PH_{suc}}$ learns to process all operations in a job before dealing with operations of another job. This could be caused by learning successor relations in the permutation of operations, which sometimes results in learning relations between operations which are not related at all (neither in the same group, nor on the same machine). On the smaller problem instances tested (results are not reported here), the summing evaluation $\mathsf{PH_{sum}}$ is improving pheromone representation $\mathsf{PH_{abs}}$. This is not the case for whizzkids97.fss. For $\mathsf{PH_{sum}}$ (on whizzkids97.fss) we observe some learning for evaporation rates 0.001 and 0.01 though, which is not the case for $\mathsf{PH_{abs}}$. Therefore we might expect $\mathsf{PH_{sum}}$ to be better than $\mathsf{PH_{abs}}$ when increasing the number of iterations.

VI. Conclusions and outlook

In this work we compared three pheromone representations from the literature with a new pheromone representation for ACO algorithms to solve FOPSSPs, a quite broad class of scheduling problems. We observed that using the new pheromone representation which leads to a learning of precedence relations between related operations is – on the problem instances chosen – resulting in a clearly improved performance in contrast to the known pheromone representations. This means that it doesn't seem beneficial to use an ACO algorithm solving a scheduling problem which is trying to learn either absolute positions in an ordered list of all operations, or learning predecessor relations in this ordered list. It is rather more beneficial to learn relative positions with regard to related operations, which is what is done in $\mathsf{PH_{rel}}$. In the future we want to investigate why the performance of the algorithm especially with the pheromone representation $\mathsf{PH_{suc}}$ is strongly decreasing at the beginning of the search process. Another aim for the future is to extend the numerical results and to particularly focus on JSSP, OSSP and MSSP instances, which are special cases of the FOPSSP.

Acknowledgments: This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential program of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] http://www.metaheuristics.net/.

[2] C. Blum, A. Roli, and M. Dorigo. HC-ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of*

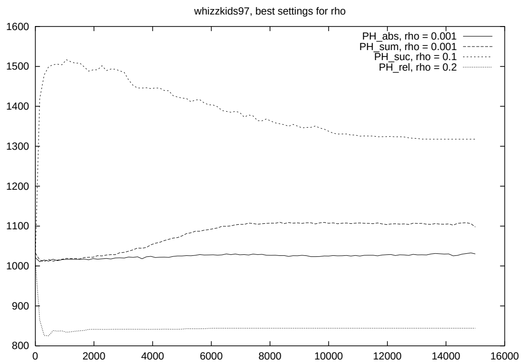

Fig. 3. The graph shows average performance for all four pheromone representations (best individual settings for ρ). The x-axis shows the number of iterations (15000), and the y-axis shows the objective function value of the best solution constructed by the ants in an iteration (averaged over 10 experiments).

MIC'2001 – Meta-heuristics International Conference, volume 2, pages 399–403, Porto, Portugal, 2001. Also available as technical report TR/IRIDIA/2001-16, IRIDIA, Université Libre de Bruxelles.

[3] P. Brucker. *Scheduling algorithms*. Springer, Berlin, 1998.

[4] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for Job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–54, 1993.

[5] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999. Also available as Technical Report IRIDIA/99-1, Université Libre de Bruxelles, Belgium.

[6] B. Giffler and G.L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[7] D. Merkle and M. Middendorf. Ant ant algorithm with a new pheromone evaluation rule for total tardiness problems. In *Proceedings of the EvoWorkshops 2000*, volume 1803 of *Lecture Notes in Computer Science*, pages 287–296. Springer, 2000.

[8] D. Merkle and M. Middendorf. On the behaviour of ACO algorithms: Studies on simple problems. In *Proceedings of MIC'2001 – Meta-heuristics International Conference*, volume 2, pages 573–577, Porto – Portugal, 2001.

[9] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice-Hall, Englewood Cliffs, 1995.

[10] A. Roli, C. Blum, and M. Dorigo. ACO for maximal constraint satisfaction problems. In *Proceedings of MIC'2001 – Meta-heuristics International Conference*, volume 1, pages 187–191, Porto – Portugal, 2001. Also available as technical report TR/IRIDIA/2001-17, IRIDIA, Université Libre de Bruxelles.

[11] B. Roy and B. Sussmann. Les problémes d'ordonnancement avec constraints dijonctives. Technical Report Note DS 9 bis, SEMA, Paris, France, 1964.

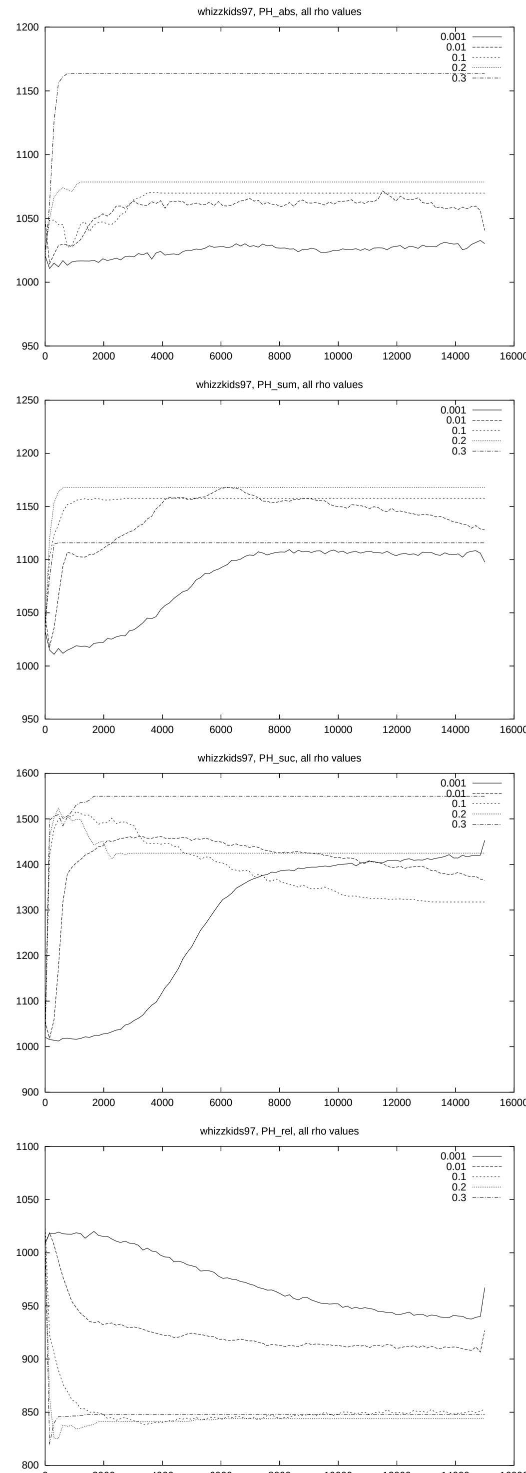

Fig. 4. The graphs show results for the 4 pheromone representations for different values of ρ. The x-axis shows the number of iterations (15000), and the y-axis shows the objective function value of the best solution constructed by the ants in an iteration (averaged over 10 experiments).