TO APPEAR ON BELGIAN JOURNAL OF OPERATIONS RESEARCH, STATISTICS AND COMPUTER SCIENCE (1994)

Ant system for Job-shop Scheduling

Alberto Colorni[*] *Marco Dorigo*[†,‡] *Vittorio Maniezzo*[¤,‡] *Marco Trubian*[*]

[*] Centro di Teoria dei Sistemi, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy, colorni@ipmel2.elet.polimi.it.

[†] IRIDIA, Université Libre de Bruxelles, Avenue Franklin Roosvelt 50, CP 194/6, 1050 Bruxelles, Belgium, mdorigo@ulb.ac.be.

[¤] Joint Research Center, EC, Ispra site, 21020 Ispra (VA), Italy, vittorio.maniezzo@cen.jrc.it.

[‡] Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract

The study of natural processes has inspired several heuristic optimization algorithms which have proved to be very effective in combinatorial optimization. In this paper we show how a new heuristic called *ant system*, in which the search task is distributed over many simple, loosely interacting agents, can be successfully applied to find good solutions of job-shop scheduling problems.

Keywords: ant system, job-shop scheduling, combinatorial optimization

1. Introduction

A recent trend in the research on combinatorial optimization heuristic algorithms takes inspiration from the observation of natural processes, modeling them in a way that can be exploited in optimization tasks. Even though the forerunners of this approach can be dated back to the 50s, with the Metropolis algorithm (Metropolis, 1953), it is only with the growth of interest on genetic algorithms (Holland, 1975; Goldberg, 1989) and on neural networks (Hopfield, Tank, 1985) that some major characteristics of this class of algorithms emerged, like distributed computation and population-based search.

Recently there has been a flourishment of nature-inspired heuristics, with several interesting proposals (see Maniezzo, Colorni, Dorigo, 1994, for an overview). Ant system (AS), a recently introduced member of this class, has already proved effective on the Traveling Salesman Problem (TSP, see Colorni, Dorigo, Maniezzo, 1991, 1992) and on the Quadratic Assignment Problem (QAP, see Colorni, Dorigo, Maniezzo, Muzio, 1994).

The AS loosely models the behavior that ant colonies have in finding food (Denebourg, Pasteels & Verhaeghe, 1983). Specifically, AS tries to capture the basic mechanisms that allow ants, which are almost blind insects, to find the shortest path between their colony and a feeding source. The algorithm distributes the search effort over many cooperating, simple agents called *ants*. These agents, taken alone, are incapable of achieving good results; on the contrary, their performance becomes very effective when they act collectively.

Cooperation goes on by updating a global memory structure, the *trail matrix* $T=[\tau_{ij}]$, that in some way memorizes structures in the search space that have been successfully exploited in the past. Each ant, at each cycle of the algorithm, constructively builds a solution on the basis of the current trail levels and of a problem-specific greedy heuristic, based on a second, global data structure, the *visibility matrix* $H=[\eta_{ij}]$.

More in detail, AS is based on the following procedure. Suppose that a solution of the problem tackled can be represented by a permutation $\pi=(\pi(1),...,\pi(n))$ of n entities which contribute to define the problem (towns in the case of TSP, activities in the QAP). Each entity is assigned to a node in a weighted digraph: the weights on the arcs are given by a pair of numbers, the first one being the trail level on the arc (whose value changes dynamically during the search process), and the second being the (fixed) visibility that assesses how good the constructive greedy heuristic rates the juxtaposition of the two adjacent nodes in the solution to build.

At the beginning of the evolution, *m* ants are randomly positioned on the nodes of the network and are free to move from a node to an adjacent one. Each ant chooses the node to move to following a Montecarlo procedure, with transition probabilities which are function of visibilities and trail levels of outgoing arcs. When it moves from node *i* to node *j*, an ant memorizes node *j* in its *tabu list*: in the following steps the ant will not be allowed to move to nodes which are contained in the tabu list, even when they are adjacent to the one the ant is currently in. Each ant k moves from node i to an adjacent node j with probability

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum\limits_{j \in \mathbf{A}_k}\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}} & \text{if } j \in \mathbf{A}_k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\mathbf{A}_k$ is the set of nodes adjacent to i in the graph, not contained in the tabu list of the k-th ant; α and β are parameters which allow the user to balance the importance given to the greedy heuristic (parameter β) with respect to the cooperation machinery based on the trails (parameter α), and t is the current iteration.

If we have m ants, we call an *iteration* of the algorithm the m moves carried out by the m ants in the interval (t, t+1); then after n iterations of the algorithm each ant has completed a permutation and each ant has filled its tabu list; we call this a *cycle* of the algorithm. After the cycle completion, each permutation is evaluated, obtaining the corresponding value of the objective function (call it L_k for the k-th ant). A value $\Delta\tau_{ij}^{k}$, which quantifies the amount of trail left by ant k on arc (i,j) during this cycle, is computed for each arc (i,j) that has been used by the k-th ant: its value is set to Q/L_k, where Q is a system parameter. The total contribution of all ants to each arc is then computed as follows:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

The trail levels to be used at the next iteration of the algorithm are given by the formula:

$\tau_{ij}(l+1) = \rho\ \tau_{ij}(l) + \Delta\tau_{ij}$

where ρ is a coefficient, such that $(1-\rho)$ can be interpreted as a trail evaporation coefficient; that is, $(1-\rho)\tau_{ij}(l)$ represents the amount of trail which evaporates on each edge (i,j) in the period between cycle *l* and cycle *l*+1.

In this paper we will show how to apply this general approach to the solution of job-shop scheduling problems. We first introduce the problem and briefly review problem-specific heuristics; then, we present the AS version that we implemented. Finally, we report computational results.

2. The Job-Shop Scheduling Problem

The Job-Shop Scheduling Problem, JSP, (Graham, Lawler, Lenstra, Rinnooy Kan, 1979) is the following. A set $M=\{M_1, \dots, M_m\}$ of machines, a set $J=\{J_1, \dots, J_n\}$ of jobs and a set of operations $O=\{u_{ij}\}$, $(i,j)\in I$, where $I\subseteq[1, n] \times [1, m]$, are given.

For each operation $u_{ij} \in O$ there is a job J_i to which it belongs, a machine M_j on which it has to be processed, and a processing time p_{ij} (a nonnegative integer) of the operation u_{ij}.

The set O is decomposed into chains corresponding to the jobs: if the relation $u_{ip} \rightarrow u_{iq}$ is in a chain, both operations belong to job J_i and there is no u_{ik} with $u_{ip} \rightarrow u_{ik}$ or $u_{ik} \rightarrow u_{iq}$.

The problem is to find a starting time S_{ij} ($\forall u_{ij} \in O$) such that it is minimized

$$\max_{u_{ij}\in O}\left(S_{ij} + p_{ij}\right) \tag{2}$$

subject to

$$S_{ij} \geq S_{ik} + p_{ik} \qquad \text{when } u_{ik} \rightarrow u_{ij} \tag{3}$$

$$(S_{ij} \geq S_{kj} + p_{kj}) \quad \vee \quad (S_{kj} \geq S_{ij} + p_{ij}) \tag{4}$$

where formula (3) expresses the operation precedence constraints on the job chains and formula (4) expresses the machine constraint saying that no more than a single job can be processed at the same time on the same machine.

The value $C_{ij} = S_{ij} + p_{ij}$ is called *completion time* for the operation $u_{ij} \in O$. The maximum of the completion times is called *makespan*.

Operations must be assigned to time intervals in such a way that no job is pre-empted; that is, once an operation is started it must be completed.

This problem, which has been studied for a long time, is known to be NP-hard (Garey, Johnson, Sethi, 1976) and has the reputation of being one of the most difficult combinatorial problems ever considered. An indication of its difficulty is given by the fact that the famous 10x10 instance (n=10, m=10) formulated for the first time by Muth and Thompson in 1963 was exactly solved only in 1989 by Carlier and Pinson using a branch and bound algorithm. Recently, other more efficient branch and bound algorithms have been proposed by Applegate and Cook (1990), and Brucker, Jurisch and Sievers (1991). However, the performance of these algorithms is very sensitive to the particular problem instance considered.

Besides exact methods, many heuristics have been developed; the most popular are *List Scheduler* algorithms which assign one operation at a time from a list ordered by some priority rule (see e.g. Panwalker, Iskander (1977) for a comprehensive survey). A more sophisticated algorithm called *Shifting Bottleneck* was given by Adams, Balas, Zawack (1988). The algorithm builds up and improves a schedule by iterative solutions of a single bottleneck machine problem. Better solutions than the ones given by deterministic algorithms were found using *Simulated Annealing* (van Laarhoven, Aarts, Lenstra, 1988; Matsuo, Suh, Sullivan, 1988) but at the cost of greater computational effort. *Tabu Search* (Glover, 1989;1990) was applied to Job-Shop by Taillard (1989), and by Dell'Amico and Trubian (1993). The problem has also been approached by a conventional *Genetic Algorithm* (Nakano, Yamada, 1991).

In order to describe our procedure for generating feasible solutions we introduce the disjunctive graph model due to Roy and Sussmann (1964).

Given an instance of JSP, we can associate with it a disjunctive graph D=(V, A, E) with V a set of nodes, A a set of conjunctive directed arcs and E a set of disjunctive (undirected) edges, defined as follows:

V= O $\{u_0\}$ [redacted] } and [redacted] $u_{N+1}\}$ are sp[redacted]al dum[redacted]ny nodes whi[redacted] identify the start and the completion of the ov[redacted]all Job-Shop)[redacted]

A= $\{(u_{ij}, u_{ij+1})$: if $u_{ij} \rightarrow u_{ij+1}$ in the [redacted]chain of job [redacted] $\forall i\}$ $\{(u_0,$ [redacted] u_{1j} is the first operation in the chain of job J_i $\forall i\}$ $\{(u_{im},$ [redacted] u_{im} is the last operation in the chain of job J_i $\forall i\}$;

E= $\{(u_{ij}, u_{hj})\ \forall i\}$.

A weight p_{ij} is associated to each vertex $u_{ij} \in$ O; vertices u_0 and u_{N+1} have weight zero. The starting time and the completion time of vertices u_0 and u_{N+1} represent respectively the starting and finishing times of the overall Job-Shop. Directed arcs represent the precedence relations (3); undirected edges represent the machine constraints (4). In Figure 1 is presented an instance with n=3 and m=4. One can see that any orientation of the edges which does not create cycles corresponds to a feasible sequencing of the operations on the machines.

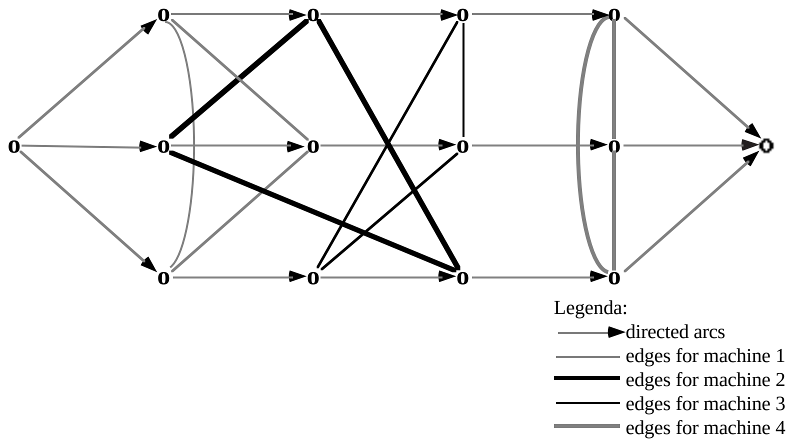


Figure 1. *A 3 jobs 4 machines JSP.*

Once the length of a path is defined as the sum of the weights of the vertices in the path, solving the Job-Shop corresponds to find an acyclic orientation of D so that the length of the longest path between u_0 and u_{N+1} (*makespan*) is minimized.

In our algorithm the visibility data structure (described in the next section) is based on the same approach used by List Scheduler algorithms. In this class of algorithms, first a rule for assigning priorities to the still unscheduled operations is defined; then, in subsequent stages, the operation with maximum priority is scheduled. Our procedure constructively generates solutions; the operations which can be feasibly assigned are those for which all predecessors are already scheduled. In this case the earliest possible starting time with respect to precedence and machine constraints is assigned to the operation with maximum priority.

A lot of different priority rules have been proposed in the literature, but none of them dominates all the others. Commonly used criteria are the following.

SPT:	select the operation with the shortest processing time;
LPT:	select the operation with the longest processing time;
SRT:	select the operation belonging to the job with the shortest remaining processing time;
LRT:	select the operation belonging to the job with the longest remaining processing time;
LRM:	select the operation belonging to the job with the longest remaining processing time excluding the operation under consideration.

In our application we adopted the LRT rule.

3. Ant s[redacted]plie[redacted] to the job-shop scheduling pro[redacted]lem

A JSP w[redacted], n jo[redacted] and operation set O, N=|O|, can be represent[redacted]in the A[redacted]by the directed, [redacted]ap[redacted] Q=<U, E>, defined as follows: U=O $\{u_0\}$[redacted]$\{(u_{ij},u_{kl})$: u_{ij},u_{kl} O} $\{(u_0,$u[redacted] is the first operation in the chain of job $J_i\}$. Node u_0 is necessary in order to specify which, among the different jobs, will be scheduled first, in case several jobs have the first operation on the same machine.

We have therefore N+1 nodes and (N(N-1)/2+n) edges, where each node, except u_0, represents the processing of a job by a machine and all the nodes are pairwise connected in both direction except u_0, which is connected only to the first operation of each job.

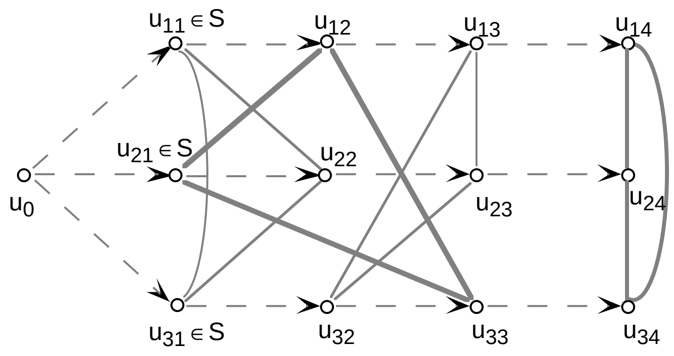
Each arc is weighted by a pair of numbers, $\{\tau_{ij}, \eta_{ij}\}$. The first, τ_{ij}, holds the trail level, while η_{ij} is an estimate of the desirability of the transition i->j according to the implemented heuristic (LRT in our case), which is therefore constant in this simple application, but possibly variable according to any specific desirability measure.

The order in which the nodes are visited by each ant specifies the directions to be given to the edges of the cliques representing each machine.

All ants are initially in u_0 and are then left free to identify a permutation of the remaining nodes. To cope with this problem, transition probabilities have to be slightly modified with respect to those computed according to formula (1): in order to have a feasible permutation it is in fact necessary to constrain the set of reachable nodes in any step not only through the tabu list, but also in a problem-dependent way. Let G denote the set of all the nodes still to be visited and S the set of the nodes whose predecessors have already been visited. Initially G=U-$\{u_0\}$ and S contains all the nodes that represent the first operation of a job. Transition probabilities are computed on the basis of formula (1), where the set of reachable nodes is equal to S. When a node is chosen, it is appended to the tabu list and deleted from G and from S; if the chosen node is not the last in its job then its immediate successor in the chain is added to S. The process is iterated until G=∅.

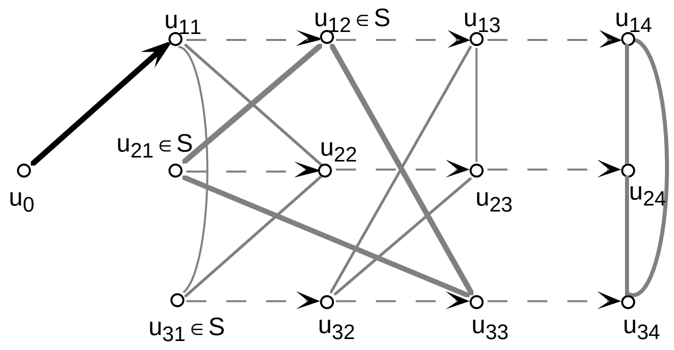
At the end, the order of the nodes in the permutation given by the tabu list specifies the orientation of the edges of the cliques representing the machines. At this point it is possible to compute the length of the critical path of the oriented graph so obtained and thus to compute the value of the objective function for the solution proposed by each ant. The trails can thus be computed in the usual way and they are laid down as specified by the AS algorithm.

The following example helps to visualize possible successive decisions taken by an ant and the solution it builds. In order to keep the example simple we used the 3x4 problem instance presented in Figure 1, and we trace the steps of a single ant which starts in u_0. At this time the reachable nodes are only the starting ones of each job.

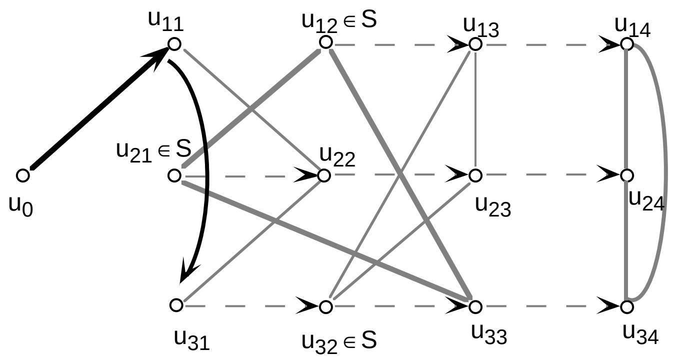


In the drawings, nodes member of S are labeled by S and the nodes in G are simply the not yet visited ones. The path followed is thickened, useless arcs and edges are removed.

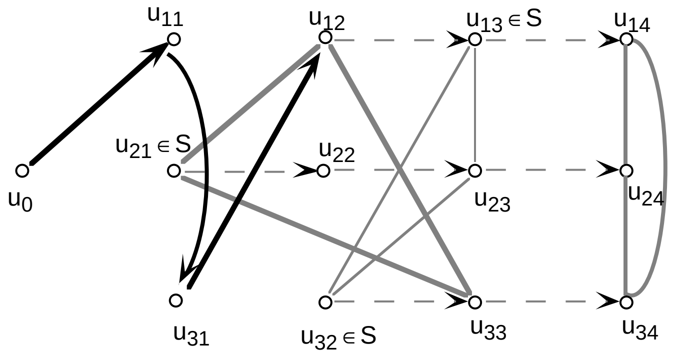
Suppose the ant chooses to go to node u_{11}; the new situation becomes the following.



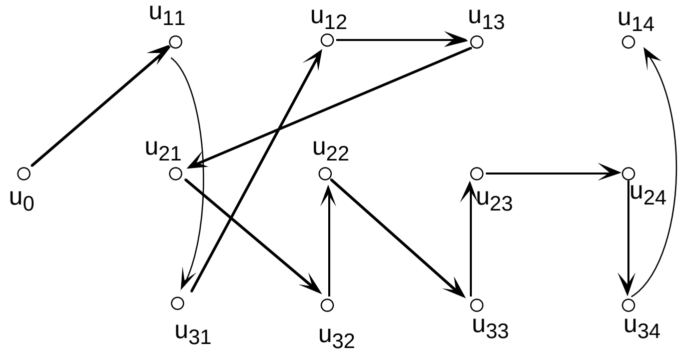
Now suppose the ant goes to u_{31}:



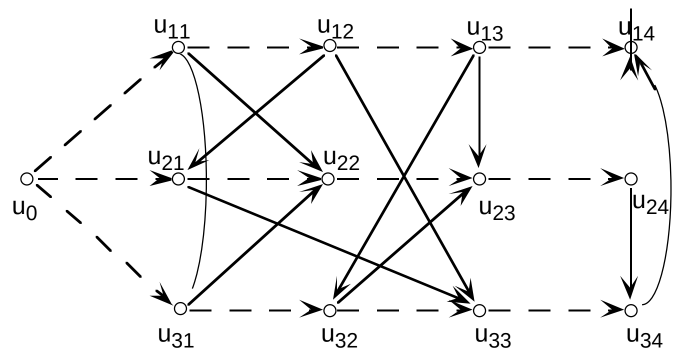
Then it goes to u_{12}, thus bringing u_{13} into S.



Suppose that the final path is:



This is a complete path: the next iteration will start the process again, with all the ants again positioned in u_0. The permutation identified is u_{11} u_{31} u_{12} u_{13} u_{21} u_{32} u_{22} u_{33} u_{23} u_{24} u_{34} u_{14}; this permutation corresponds to the following feasible edge directioning.



4. Computational results

In this section we present the results of experiments about the AS parameter setting and the AS performance on problems with different degrees of complexity.

4.1. Best values of the AS parameters

The parameters considered here are those that affect directly or indirectly the computation of the probability in formula (1): α, β, ρ[1]. The number of ants has always been set to be equal to the number n of jobs and the priority rule has always been LRT. We tested several values for each parameter; we set the default value of the parameters at α=1, β=1, ρ=0.7, and in each experiment only one of the values was changed. The values tested were: $\alpha \in \{0, 0.5, 1, 5, 10\}$, $\beta \in \{0, 0.5, 1, 5, 10\}$ and $\rho \in \{0.3, 0.5, 0.7, 0.9\}$. The tests reported here are based on a standard 10x10 problems, ORB4, originally proposed by Applegate and Cook (1990), which has an optimal solution of 1005. All the tests have been carried out for 3000 iterations and were averaged over 5 trials.

Beside the path length, we were interested also in investigating the *stagnation* behavior, that is the situation in which all the ants make the same path. When stagnation occurs the AS does not explore new solutions, and therefore the best path achieved before stagnation will not be improved any more. We experimentally found that only very particular parameters settings made the system enter the stagnation behavior; in particular, we found that high values of α favor stagnation.

The best and the average results of 5 trials for each parameter setting, obtained testing different values of α, β and ρ, are respectively presented in Figures 2, 3 and 4. The obtained results show that α and β have an optimal range around 1, ρ around 0.7.

[1] Previous experiments (Colorni, Dorigo, Maniezzo, 1991) have shown that quantity Q does not influence the algorithm performance.

The algorithm mainly uses the greedy heuristic to guide search in the early steps, but as the computation runs it starts exploiting the global information contained in the values τ_{ij} of trail. This explains the value ρ=0.7: the algorithm needs to have the possibility to forget part of the experience gained in the past in order to better exploit new incoming global information.

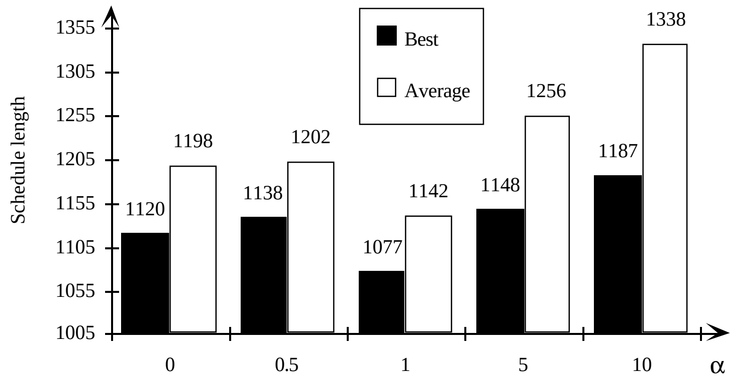


Figure 2. *Best and average (5 trials) schedule length with increasing values of α.*

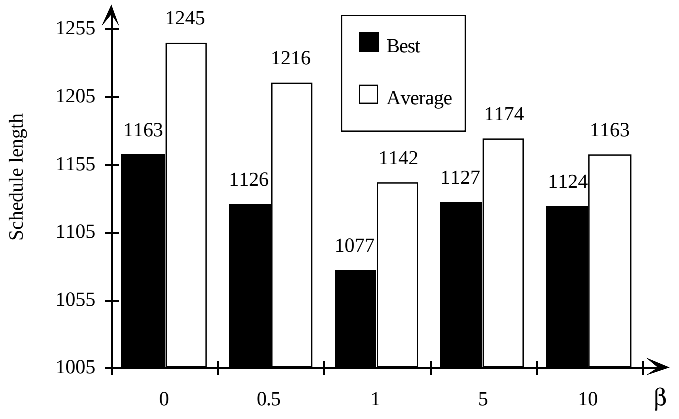


Figure 3. *Best and average (5 trials) schedule length with increasing values of β.*

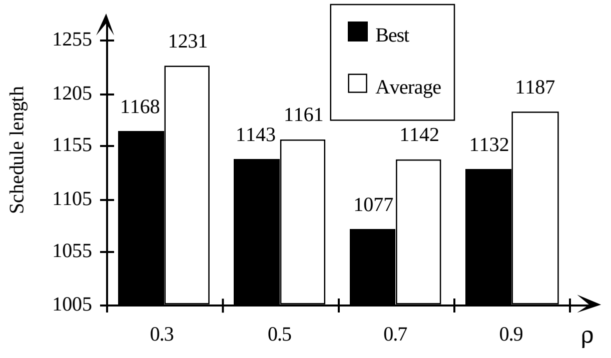


Figure 4. *Best and average (5 trials) schedule length with increasing values of ρ.*

The main strengths of the Ant-cycle algorithm can be summarized in the following points.

- With the best parameter values the algorithm always finds good solutions (the best one about 7% from the best-known value of 1005).
- The algorithm quickly finds satisfying solutions; nevertheless it does not enter the stagnation behavior, and the ants continue to search for new, possibly better, solutions.
- We tested the algorithm on problems with increasing dimensions, and we found the parameter sensitivity to the problem dimension to be very low.

4.2. Experiments on different problem instances

A second set of experiments was run in order to assess the effectiveness of our algorithm on problems of increasing difficulty. In this phase we were not interested in shaping an algorithm that provided the best possible solutions, an activity that requires a careful study of the greedy heuristic and of the neighborhood structure used, but in verifying the effectiveness of the approach even with the rude heuristic we used. We plan in fact for the future to investigate the possibilities offered by the AS when used as a meta-heuristic, specifically as a global optimization algorithm that explores the search space in order to provide good starting points for local optimization; the research we report here is aimed at the validation of the AS approach as a viable, effective heuristic for JSP problems. To do that, we first tested it on a "easy" instance, MT06 (a 6x6 problem originally proposed by Muth and Thompson (1963)), finding the best-known within a few iterations in all the runs (the experiment was repeated 5 times). Then we attacked tougher problems: three 10x10 (MT10 by Muth and Thompson (1963); ORB1 and ORB4, by Applegate and Cook (1990)) and a 10x15 (LA21, by Lawrence (1984)) problems.

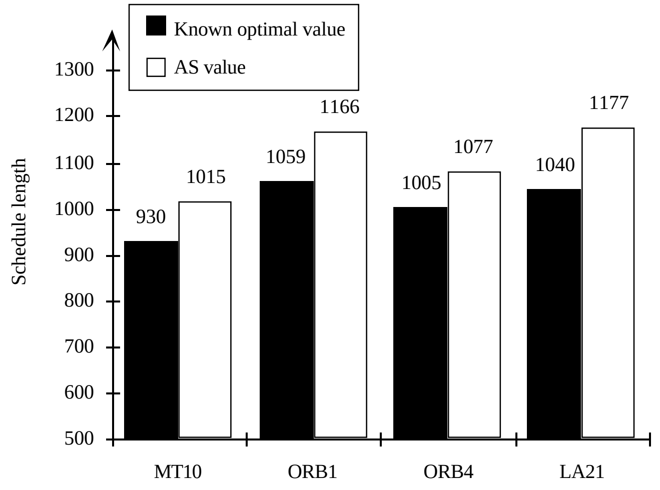


Figure 5. *Best results obtained for different problems.*

In three out of four cases, we identified solutions within 10% from the optimum, as shown in Figure 5. These results represent the best value achieved over five runs, with the best parameter setting (α=1, β=1, ρ=0.7) presented before: we never got the optimum result, but a quick convergence to satisfying solutions was maintained for all the problems.

All the runs were stopped after 3000 iterations, but the search was still actively going on, as was testified by the non-zero variance of the set of solutions proposed by the ants, and as it is suggested by the plot of the best solution found at each iteration, presented in Figure 6 for the ORB4 problem.

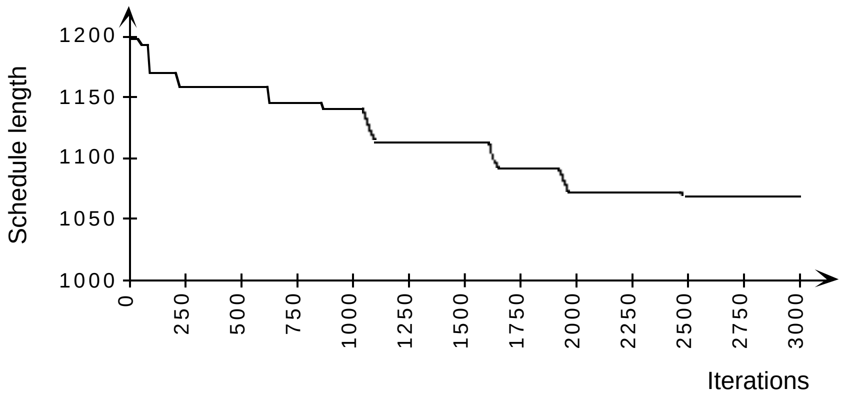


Figure 6. *Evolution of best solution achieved in the ORB4 problem.*

5. Conclusions

In this paper we presented an original heuristic called *ant system*, and we applied it to the job-shop scheduling problem. AS is a stochastic optimization algorithm that takes inspiration from the model of a natural process. One of its more attractive characteristics is the distribution of search over several simple, loosely interacting agents called *ants*. The effectiveness of the algorithm is due to the cooperation among the ants, which takes place periodically by modification of a global problem-specific memory structure, the *trail matrix*.

The effect of applying AS to a JSP problem is a probabilistic superposition of effects: each ant, if isolated (i.e., if α=0), would move according to a local, greedy heuristic. This greedy heuristic guarantees only locally optimal moves; it doesn't give good results because greedy local improvements very often lead to very bad final steps. In fact, the tour followed by an ant ruled by a greedy policy is composed by some parts that are very good and some others which are not. If we now consider the effect of the simultaneous presence of many ants, then each ant contributes to a part of the trail distribution; good parts of paths will be followed by many ants and therefore they receive a great amount of trail, which in turn will make these parts more attractive (thus giving rise to a positive feedback – or *autocatalytic* – process); bad parts chosen because obliged by constraints satisfaction (the tabu list) will be chosen only by few ants and therefore their trail level remains low. When agents interact, the autocatalytic process converges on good solutions very quickly, without getting stuck in local optima.

The AS has already proved effective in solving TSP (Colorni, Dorigo, Maniezzo, 1991; 1992) and QAP (Colorni, Dorigo, Maniezzo, Muzio, 1994). The application reported here further suggests the robustness of the approach, showing how it is one of the most easily adaptable population-based heuristics so far proposed and how its basic computational paradigm (the updating of a global problem representation by many simple agents) is indeed effective under very different conditions.

References

Adams J., Balas E., Zawack D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science.* 34, 391-401.

Applegate D., Cook W. (1990). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*. 3, 149-156.

Brucker P., Jurisch B., Sievers B. (1991). A Fast Branch & Bound Algorithm for the Job-Shop Scheduling Problem. *Internal Report 136*, Fachbereich Mathematik/Informatik, Universität Osnabrück.

Carlier J., Pinson E. (1989). An algorithm for solving the Job-Shop problem. *Management Science*. 35, 2, 164-176.

Colorni A., Dorigo M., Maniezzo V. (1991). Distributed Optimization by Ant Colonies. *Proceedings of ECAL91 - European Conference on Artificial Life*, Paris, France, Elsevier Publishing, 134–142.

Colorni A., Dorigo M., Maniezzo V. (1992). An Investigation of some Properties of an Ant Algorithm. *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN 92)*, Brussels, Belgium, Elsevier Publishing, 509–520.

Colorni A., Dorigo M., Maniezzo V., Muzio L., (1994). Il sistema formiche applicato al problema dell'assegnamento quadratico (in Italian). Submitted to *Ricerca Operativa*.

Dell'Amico M., Trubian M. (1993). Applying Tabu-Search to the Job-Shop Scheduling Problem. *Annals of Operations Research*, 41, 231-252.

Denebourg J.L., Pasteels J.M., Verhaeghe J.C. (1983). Probabilistic Behaviour in Ants: a Strategy of Errors?. *Journal of Theoretical Biology 105*, 259-271.

Garey M.R., Johnson D.S., Sethi R. (1976). The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* 1, 117-129.

Glover F. (1989). Tabu Search, Part I. *ORSA Journal on Computing*, 1,3, 190-206.

Glover F. (1990). Tabu Search, Part II. *ORSA Journal on Computing*, 2,1, 4-32.

Goldberg D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.

Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics,* 5, 287-326.

Holland J.H. (1975). *Adaptation in Natural and Artificial Systems,* Ann Arbor: The University of Michigan Press.

Hopfield J.J., Tank D.W. (1985). Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics 52*.

van Laarhoven P.J.M., Aarts E.H.L., Lenstra J.K. (1992). Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40, 113-125.

Lawrence S. (1984). Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *GSIA*, Carnegie Mellon University.

Maniezzo V., Colorni A., Dorigo M. (1994). ALGODESK: An experimental comparison of eight evolutionary heuristics applied to the QAP problem. To appear in *European Journal of Operational Research.*

Matsuo H., Suh C.J., Sullivan R.S. (1988). A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem. *Working paper 03-44-88,* Graduate School of Business, University of Texas, Austin.

Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., Teller E. (1953). Equation of State Calculations by Fast Computing Machines. *J. of Chemical Physics*, 21, 1087-1091.

Muth J.F., Thompson G.L. (1963). *Industrial Scheduling*. Englewood Cliffs, N.J., Prentice Hall 1963.

Nakano R., Yamada T. (1991). Conventional Genetic Algorithm for Job Shop Problems. *Proc. of the 4th Int. Conference on Genetic Algorithms*, San Diego, California, 474-479.

Panwalker S.S., Iskander W. (1977). A survey of Scheduling Rules. *Oper. Res.* 25, 1, 45-61.

Roy B., Sussmann B. (1964). Les Problemes d'Ordonnancement avec Contraintes Disjonctives. *Note DS n.9 bis*, SEMA, Montrouge.

Taillard E. (1989). Parallel Tabu Search Technique for the Jobshop Scheduling Problem. *Internal Report ORWP 89/11*, Departement de mathematiques, Ecole Polytechnique Federale de Lausanne, Lausanne.