Transactions on Information and Communications Technologies vol 8, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

How to schedule a flow shop plant by agents

T. Daouas, K. Ghedira, J.P. Muller
Institute of Computer Science and Artificial Intelligence, Emile Argand 11, 2007 Neuchatel, Switzerland

Abstract

The flow shop scheduling problem consists, according to a certain number of criteria, in finding the best possible allocation of n jobs on m resources, so that operations of every job must be processed on all resources in a unique order. Because of its highly combinatorial aspect, this scheduling procedure has been widely studied in the literature by exact and mostly heuristic methods. The approach, we adopt here to deal with this problem, combines a Multi-Agent system with a stochastic combinatorial optimization tool, the simulated annealing. This paper stresses the efficiency and the optimality of a distributed implementation of this tool compared to a classical one.

1 Introduction

Scheduling is very important in manufacturing systems. Because of its NP-Complete aspect, its combinatorial character, its dynamic nature and its practical interest for industrial applications, scheduling, generally speaking and more exactly in a plant [5], has been widely studied in the literature using exact and mostly heuristic methods.

Among these methods, some ones based on constraint propagation techniques, for instance ISIS [5], SOJA [15], etc. A CSP, for Constraint Satisfaction Problem, perspective has also been proposed in [6] and used by [17] to modelize the scheduling problem in terms of variables involved in constraints, the solution being a variable's instanciation satisfying the set of constraints. Moreover, stochastic methods providing sub-optimal solutions have been suggested; among these ones, we name [16] which uses SA techniques, [2] which uses Taboo search and [14] which bases its search on genetic algorithms. Scheduling has also been dealt with as a Constrained Optimization Problem (COP) in the CORTES approach [7], which is a factory scheduling system that solves resource allocation COP's using Constrained Heuristic Search.

Despite the variety of the methods just mentioned, scheduling problem remains difficult to solve, justifying us to explore a new direction: Multi-Agent

Systems (MAS), for instance the Eco-problem solving [4], an approach based on interactions between agents, each of them trying to reach its own satisfaction. In this framework, scheduling has been formulated by Sycara and Liu [17] as a Distributed CSP. An agent has to instanciate a sub-set of variables assigned to it. Constraints may exist between agents. A solution is reached when all constraints intra-agents and inter-agents are satisfied. Operations represent variables, their possible values represent reservations (start time and required resources) and both precedence relations and resource capacities represent constraints.

A similar approach combining MAS with Simulated annealing (SA) has been proposed for the resource allocation problem [12]. The underlying efficient revision mechanism to deal with the dynamic aspect [13] [8] as well as the good experimental results [12] encouraged us to extend this approach to the flow shop scheduling problem. Moreover, this approach has been successfully adapted to both static aspect [9] [11] and dynamic aspect of CSP [10].

After describing the problem we deal with, we introduce a model based on the combination of MAS and SA [3]. Then, comparisons between both distributed and classical centralized implementations of SA, will be done thanks to examples randomly generated.

2 The Problem

Given a set of jobs attached, each one, to its process plan (directed graph of operations), its quantity (number of components to be produced) and its deadline, and given a physical model consisting of resources separated by stock stations, the scheduling here consists in allocating each operation to possible resources taking into account the precedence relations existing between operations and the deadlines of jobs in order to provide the best possible sequence of jobs.

From now on, and given an operation, we call its previous (resp. next), the operation being directly before (resp. after) with respect to the partial order defined by the process plan. We call its precedent (resp. following), the operation which will be executed directly before (resp. after) on the same resource.

In the framework of the flow shop scheduling problem, we adopt the following hypotheses:

- The graph structure of resources and stock stations representing the physical model is reduced to a linear one. Consequently, the process plan becomes a linear sequence of operations. These operations have strict precedence relations between each other, that is to say, if A_{i+1} is the next operation to A_i, execution of A_{i+1} cannot start before the completion of, at least, one component of A_i.
- A stock station can have only two possible states. Given two resources R_1 and R_2 separated by a stock station SS_{12}, if R_1 is faster than R_2 then SS_{12} is assumed to be full otherwise it is empty.

To optimize the sequence of jobs, we choose the following criteria:

- Minimizing the setup duration for each resource. The setup duration is the time required by a resource to change tool. Instead, each operation needs one or more specific tools to be performed, so a time setup may exist between two successive operations on the same resource.
- Maximizing the slack time of each job. A slack time is the difference between the deadline and the completion time of the job.

More formally, we are concerned with maximizing a cost function f such that:

$$f = -\alpha * (nb\_setup * dur\_setup) + \beta * slack$$

- α and β are weighting parameters provided by the user,
- *nb_setup* is the total number of setups,
- *dur_setup* is the setup duration and
- *slack* denotes the sum of slack times of all jobs.

3 The Multi-Agent_Simulated Annealing model

3.1 The Simulated Annealing tool

Coming from statistical physics, SA allows to avoid as much as possible the local optima trap which often occurs with greedy algorithms. Let f be a cost function to maximize, SA consists in:

- allowing cost function deterioration, hoping to improve it later,
- controlling this possibility with a stochastic process that is guided by a parameter, the tolerance, which is usually called the temperature: Initially, there is a high tolerance to cost function deterioration, and progressively this tolerance decreases until the system reaches the equilibrium state. $T(k)$ denotes the tolerance at time k, the probability to accept the new state g once generated from the current state c is equal to:

$$A_{cg} = \begin{cases} 1 & if\ f(g) \geq f(c) \\ e^{\frac{f(g) - f(c)}{T(k)}} & Otherwise \end{cases}$$

3.2 The Multi-Agent approach

According to the original approach [13], each agent has acquaintances (i.e. the agents that it knows and with which it can communicate) and a behavior based on the search of its satisfaction: If satisfied then satisfaction-behavior else unsatisfaction-behavior.

The proposed model involves two types of agents: Job agents and Resource agents. Before detailing these agents, let us introduce briefly the global dynamics:

Each job requests the resources by sending its operations one by one, according to the precedence relations between these operations. For each job, we associate a rank representing its position on the resources. This rank is determined randomly when the first operation is allocated. When a resource is requested by a job, its behavior, based on the SA, consists of two phases: a) Generation of a new state, comprising necessarily the new operation just received, and b) decision on whether to accept or to reject this new state.

3.2.1 The Job agents A job has as acquaintances all the resources. It is satisfied when all its operations are allocated, and in that case it does nothing. In the unsatisfaction case, its behavior consists in sending to resources its operations not yet allocated as follows:

- If A_i is allocated, the operation to be sent is A_{i+1}.
- If A_i is refused by a resource, the job it belongs takes all its operations out of the resources, and the next candidate operation will be A_1.
- If A_i had been already allocated to a resource and has been rejected later on because of the overstepping of the deadline, A_i remains the next candidate operation.

3.2.2 The Resource agents A resource has as acquaintances all the jobs. It is satisfied when it is not requested by any Job. Whenever a resource is requested by a job, its unsatisfaction behavior based on SA consists in

generating a new state, comprising necessarily the incoming job, and deciding whether to accept it or not. In addition, the more the cost function of a resource is high the more a resource is satisfied; consequently, if the tolerance is still greater than zero and all jobs are allocated, an improvement phase is performed hoping to increase the cost function.

• New state generation: The operation just received is attached with both the deadline and the earliest completion time of its job. This information helps the resource to check the deadline overstepping. The earliest completion time of the job is determined without taking into account neither the setup times nor the transfer times between operations. To allocate this operation, the resource behavior consists in:

- attributing a rank to the corresponding job,
- determining the start and finish dates of the operation,
- possibly ejecting other jobs already allocated to make sure that the job in question does not overstep its deadline, and
- reacting to side effects by ejecting operations that overstep their deadlines because of the new operation allocation. We notice here that ejecting an operation implies automatically ejecting all its next operations allocated to next resources (precedence relations).

• Decision: If the new generated state is accepted, every ejected job takes all its operations out of the other resources. In the same way, every job having an operation ejected due to the side effects reaction, takes all the next operations out. However, if the new generated state is refused, the resource informs the job, which has requested it, of its decision. Therefore this job takes all its operations out and tries, later on, to be allocated to another position.

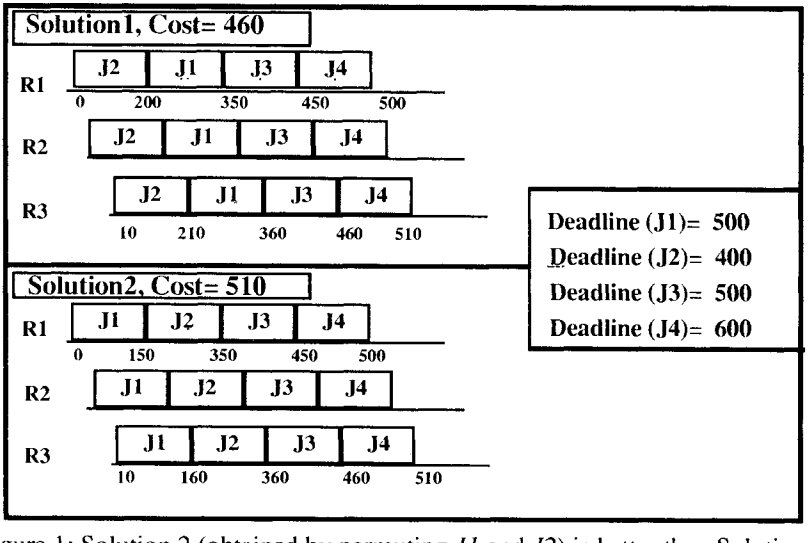

Figure 1: Solution 2 (obtained by permuting *J1* and *J2*) is better than Solution 1.

• Improvement phase: Even if all jobs are allocated, the obtained sequence is not necessarily optimal. Therefore we add an improvement phase at the level of the first resource R_1 which is performed as follows: Resource R_1 ejects randomly one job, to allow the system to visit more possible configurations (figure 1).

This phase is repeated while the tolerance is greater than zero. We will come back later to that parameter in the following section. The choice of R_1 is justified by the fact that all process plans force any job to first request R_1.
Let us remark that for some hard problems, the model cannot find a configuration satisfying all deadlines. Consequently, the final configuration may have some jobs not allocated. For such jobs, the system asks via the Machine agent (see next paragraph) the user for possibly relaxing some deadlines.

4 Implementation of simulated annealing

4.1 The machine agent

For the need of implementation, a third class of agents, which contains only one agent called the Machine agent, has been added. It has as acquaintances the set of jobs and the set of resources. It is satisfied when all the jobs are allocated. When it is satisfied, it gives the best configuration found so far during the process. If it is unsatisfied, it does nothing. We notice that this agent does not interfere in the problem resolution process, i.e., in the dialogue between Job and Resource agents. It is only an intermediary between the agents society and the user.

4.2 Two alternatives

It has been proved in [1] that the SA algorithm converges asymptotically to a sub-set of optimal solutions. Unfortunately, this convergence requires an infinite run-time. Consequently, we propose two alternatives to put SA in practice:
• The first one, CSA, consists in implementing a unique centralized SA (as used generally in literature) whose parameters, namely the tolerance and its decreasing scheme, are shared and controlled by all the Resource agents.
• The second one, DSA, consists in fully exploiting the Multi-Agent philosophy, namely locality and autonomy, by using a Distributed SA. In this case, each agent maximizes its local cost function, which is the restriction of the global function to its environment (operations allocated on it). It controls the above mentioned parameters of its own SA according to its knowledge and its interactions with its acquaintances.

In order to make fair comparisons between these two alternatives, we have been inspired by [12] to determine the following decreasing scheme, also called in the SA literature *cooling scheme*.
A decreasing scheme consists of an initial tolerance, a decreasing function and a termination criterion.
• For the first alternative, the initial tolerance is equal to the sum of all job slack times (for each job, the slack time here, is the difference between its deadline and its earliest completion time, such that its start time is zero). The tolerance is decreased after m cost deterioration acceptances, where m is the number of Resource agents. The termination occurs when the tolerance reaches the zero level.
• For the second alternative, the initial tolerance of each Resource agent is also equal to the sum of all job slack times. Each Resource agent decreases its tolerance in the same manner as CSA whenever it accepts a local cost deterioration. The termination occurs when the tolerance of the first resource R_1 reaches the zero level. The choice of R_1 is due to the same reason as in paragraph Improvement phase.

4.3 Experimental comparisons

4.3.1 Experiment design The idea is to find the best finite-time implementation of the SA algorithm resulting in a compromise between a good quality of solution (the nearest to optimal solutions) and an acceptable run-time. For this purpose, the two alternatives CSA and DSA are compared in terms of optimality by measuring the rate of optimality, and efficiency by measuring the run-time. This run-time includes the time spent to exchange messages between agents.

The test examples have been randomly generated as follows: we have fixed the number of jobs to *5* and attributed to the number of resources the following values *{2, 4, 6, 8, 10}*, and vice versa. For each job, the deadline is equal to *Random [1.1, 1.3]* times the sum of:

- the flow time of the job, which is its execution duration without neither setup nor transfer duration and
- an approximate duration of the other jobs.

The number of components of each operation (the same for all operations belonging to the same job), is chosen randomly between *1* and *200*.

Moreover, and considering the random aspect of SA, we have performed *10* runs per example and taken the average of run-time. The rate of optimality is computed as follows: Let *max* denotes the maximal value reached during ten runs and let *p_max* be the number of runs where *max* is reached, the optimality rate is determined by: *(p_max/10) * 100*

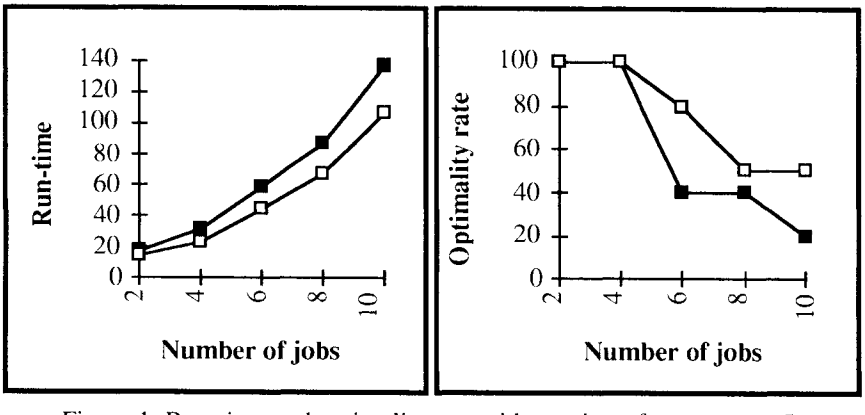

Figure 1: Run-time and optimality rate with *number of resources* = 5.

4.3.2 Experimental results The efficiency is represented by the run-time, and the optimality by the optimality rate which reflects the quality solution.

The results are shown in figures where black squares represent CSA and white squares represent DSA. The maximum cost values reached by the two mechanisms are the same, but the rate is not the same as we will see later.

At constant number of resources equal to *5*, figure 2 shows that, at the same optimality rate, DSA is better than CSA from the run-time point of view. It requires, indeed, less run-time. This advantage becomes more pronounced as the number of jobs increases; indeed, DSA needs less time to have a better optimality rate than CSA. At constant number of jobs equal to *5*, figure 3 allows to keep the same remarks as before.

To summarize, we can assert that in terms of run-time, DSA always outperforms CSA. In terms of optimality, DSA and CSA have the same rate for numbers of jobs less than numbers of resources.
When the number of jobs oversteps the number of resources, the optimality rate of DSA becomes much better. This result is very important since, in practice, problems with large numbers of jobs seem to be the most difficult according to our selected criteria. Such advantages become more important with an implementation of DSA on parallel architecture.

Moreover, even if we slow down the decreasing scheme of DSA, its performances would be, at least, as good as those of CSA. On the other hand, even if we accelerate the decreasing scheme of CSA, its performances would be either worst or, at most, the same as those of the DSA. Consequently, DSA would give in both cases a better complexity-quality compromise.

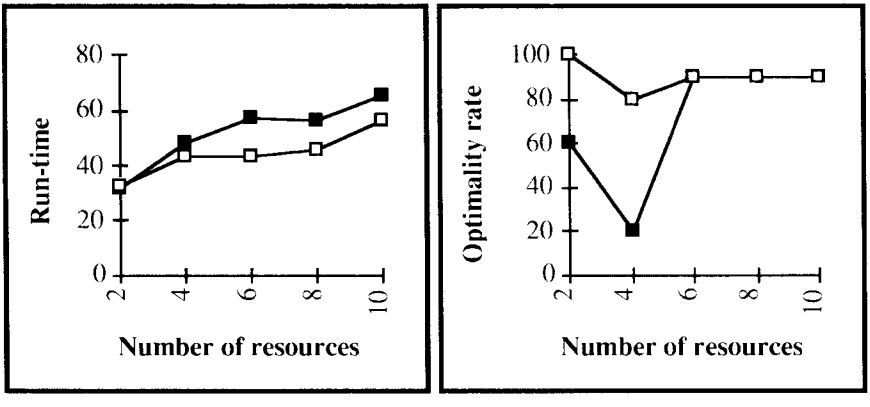

Figure 2: Run-time and optimality rate with *number of jobs* = 5

5 Conclusion and perspectives

In this paper, the flow shop scheduling problem has been treated by a Multi-Agent model consisting of Job agents and Resource agents in interaction, trying to find the best possible sequence of allocated jobs. For that purpose, an optimization tool, the SA, has been implemented at the level of each Resource agent.

Experimental comparisons, based on randomly generated examples, with a centralized implementation of this tool have shown its power in terms of efficiency and optimality.

As far as our future work is concerned, we have to do on the one hand further experimentation with more complex examples, and on the other hand comparisons of the model with other known methods.
Moreover, we intend to extend this model to the general case of the job shop scheduling problem.

Acknowledgements

This work is supported by the Swiss National Founds of the Scientific Research under contract # *SPPIF5003-034319*.

References

1. Aarts, E.H.L. & Van Laarhoven, P.J.M. *Simulated annealing: Theory and applications*, Reidel, D. Publishing Company, 1987.
2. Amico, D. A. & Trubian, M. Applying Taboo search to the job shop scheduling problem, *Annals of Operations Research*, Vol.41, 1-4, 1993.
3. Daouas, T. & al A Distributed approach for the flow shop scheduling problem, *Proceedings of the Int. Conf. of AI Applications*, Cairo, 1995.
4. Ferber, J. & Jacopin, E. *The framework of Eco-problem solving, Decentralized Artificial Intelligence*, Vol.2, Editions North Holland, 1990.
5. Fox, M. S. Isis: A constraint-directed reasoning approach to job shop scheduling, *Proceedings of IEEE'83*, 1983.
6. Fox, M. S. & Sadeh, N. Why scheduling difficult? a CSP perspective, *Proceedings of ECAI'91*, 1991.
7. Fox, M. S. & Sycara, K. P. Overview of CORTES: A constraint based approach to production planning, scheduling and control, *Proceedings of the 4th Int. Conf. on Expert systems in Production and Operations Management*, 1990.
8. Ghedira, K. A reactive and distributed approach to the revision problem in the framework of resource allocation problem, *Proceedings of IEEE-ETFA*, Melbourne, 1992.
9. Ghedira, K. A Distributed approach to partial constraint satisfaction problems, *Proceedings of MAAMAW'94*, Odense, 1994.
10. Ghedira, K. Distributed simulated re-annealing for dynamic constraint satisfaction problems, *Proceedings of TAI'94*, 1994.
11. Ghedira, K. Partial constraint satisfaction by a Multi-Agent_Simulated annealing approach, Vol.1, *Proceedings of the Int. Avignon Conf. on AI*, 1994.
12. Ghedira, K. & Verfaillie, G. Approche Multi-Agents pour les problèmes d'affectation, *Proceedings of the Int. Avignon Conf. on AI: Expert systems and their applications*, Avignon, 1991.
13. Ghedira, K. & Verfaillie, G. A Multi-Agent model for the resource allocation problem: a reactive approach, *Proceedings of ECAI'92*, Vienna, 1992.
14. Lawrence, D. Job shop scheduling with genetic algorithms, pp. 136-140, *Proceedings of the 1st Int. Conf. on Genetic algorithms and their applications*, 1985.
15. Lepape, C. & al Soja: Un système d'ordonnancement opportuniste, *Proceedings of the 5th Int. Avignon Conf. on AI*, Avignon, 1985.
16. Sridhar, J. & Rajendran, C. Scheduling in a cellular manufacturing system: a simulated annealing approach.
17. Sycara, K. P. & Liu, J. Emergent constraint satisfaction through Multi-Agent coordinated interaction approach, *Proceedings of the 5th European workshop on Modelling autonomous agents in a Multi-Agent world*, 1993.