CENTRALIZED AND DECENTRALIZED OPTIMISATION TECHNIQUES FOR THE FLEXIBLE JOB SHOP SCHEDULING PROBLEM

Meriem Ennigrou[1] and Khaled Ghédira[2]
1 ur. SOIE, Stratégies d'Optimisation des Informations et de la connaissancE
IPEIM, Institut Préparatoire aux Etudes d'Ingénieurs – ElManar, 2092 El Manar 2 BP 244, Tunisie

2 ur. SOIE, Stratégies d'Optimisation des Informations et de la connaissancE
ENSI, Ecole Nationale des Sciences de l'Informatique, 2010 Campus Universitaire la manouba, Tunisie

Keywords: Multi-Agent System, flexible Job Shop, Scheduling, Tabu Search.

Abstract: This paper proposes two Multi-agent approaches based on a tabu search method for solving the flexible Job Shop scheduling problem. The characteristic of the latter problem is that one or several machines can process one operation so that its processing time depends on the machine used. Such a generalization of the classical problem makes it more and more difficult to solve. The objective is to minimize the makespan or the total duration of the schedule. The proposed models are composed of three classes of agents: Job agents and Resource agents and an Interface agent. According to the location of the tabu search core, two versions have been proposed. The first one places the optimisation method only on the Interface agent whereas the second associates to each Resource agent its own optimisation process.

1 INTRODUCTION

In the later decades, extensive researches on scheduling have been reported from both theoretical and practical issues. Scheduling means allocating a set of jobs to a finite set of resources over time while satisfying a set of constraints. An important goal in the scheduling function is to assure that the work is completed as early as possible.

Among the most difficult scheduling problems, we find the *Job Shop Scheduling Problem (JSSP)*. Finding an optimal solution for such problems in a reasonable time seems to be very hard, in the majority of cases, because of their high complexity. In fact, this problem falls into the category of NP-hard problems for which exact solving methods are inappropriate since they explode with problem size. However, approximate methods are more suitable for such problems. The latter are either based on local search techniques such as tabu search and simulated annealing or on evolutive techniques such as genetic algorithms and ant systems.

In this paper, we are concerned with an extended class of the JSSP, namely the *flexible Job Shop (FJSP)*, to which we propose two Multi-Agent models based on the tabu search optimisation method. The objective is to minimize the makespan or the total duration of the schedule.

2 THE FJSP

A JSSP consists in performing a set of n jobs $\{J_1, \ldots, J_n\}$ on a set of m resources $\{R_1, \ldots, R_m\}$. Each job J_i, $i=1,\ldots,n$, is composed of n_i operations that must be performed on the different resources according to a predefined order, known as the *job process routing*. This one characterizes the *precedence constraints* existing between the operations of one job. In addition, each operation has a *processing time* known in advance and can be processed by only one resource.

Furthermore, each job has to be achieved in a temporal range defined by its *release date*, before which the job cannot be started, and its *due date*, before which the job must be completed. This temporal range defines the *temporal constraints* of that job. Moreover, a resource can perform only one operation at a time, which corresponds to the

Ennigrou M. and Ghédira K. (2005).
CENTRALIZED AND DECENTRALIZED OPTIMISATION TECHNIQUES FOR THE FLEXIBLE JOB SHOP SCHEDULING PROBLEM.
In *Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics - Robotics and Automation*, pages 31-36
DOI: 10.5220/0001176900310036

disjunctive constraints, and an operation cannot be interrupted unless it is finished, i.e. no *pre-emption* is allowed. A solution for the JSSP consists in fixing a start time for each operation satisfying the set of constraints.

FJSP, first introduced by (Nuijten & Aarts, 1996), is a generalisation of the above mentioned problem, where each operation can be processed by more than one resource and has consequently a processing time depending on the resource used. A solution consists then not only in sequencing the operations on the resources and fixing them a start time but also in allocating them to a resource likely to achieve them. This problem is also NP-hard.

3 TABU SEARCH

The models we propose in this article are based on a combinatorial optimisation technique, namely the *tabu search* (TS) method, proposed by (Glover, 1986), which is a meta-heuristic based on the local search principle. Beginning from an initial solution, the local search consists to choose, at each iteration, the best solution in the current solution neighbourhood, even if it does not improve the quality of the solution. A neighbourhood is composed of all the solutions obtained by a simple move on the current solution. These solutions are named, then, *neighbours* of the current one. TS have proved its power to handle JSSPs. Several researches have used TS and good results have been obtained. Among the approaches proposed for the JSSP, the ones proposed by (Mastrollili & Gambardella, 2000), (Brucker & Neyer, 1998) and (Chambers & Barnes, 1996).

In order to escape local optima in which the system can be easily trapped, TS uses a temporary memorisation structure in which it keeps track of the last visited solutions: the *tabu list*. In fact, a solution is forbidden during a number of iterations equal to the tabu list size. Then, the best solution among the ones not forbidden is selected for the next iteration.

Although its efficiency in solving many difficult problems, TS remains yet hardly adaptable to FJSP because of the great number of parameters to define:

- initial solution,
- neighbourhood function,
- evaluation of the current solution,
- tabu list size, etc.

Later in this paper, we will describe briefly our adaptation of the different parameters to the FJSP. The next section presents the two multi-agent models proposed and subsequently their global dynamic.

4 MULTI-AGENT MODELS

Two Multi-Agent models have been proposed for solving FJSP. The first one consists in centralizing the optimisation process in a unique agent responsible for finding the optimal solution in cooperation with the remainder of the agents which are responsible for generating successive feasible solutions at each step of the process. Whereas, the second one distributes the optimisation process between a collection of agents cooperating together in order to find the best possible solution.

Since scheduling problems involve two sorts of constraints: the ones concerning the jobs, namely precedence and temporal constraints, and those concerning the resources, namely disjunctive constraints, both Multi-Agent models proposed in this paper are then composed of two agent classes: *Job Agents* and *Resource Agents* responsible for the satisfaction of the two classes of constraints. In addition, a third agent class, containing a single agent, the *Interface agent*, is added to both models. The degree of importance of the latter agent in the solving process differs from the first model to the second. In fact, in the first model, it contains the core of the TS method. However, in the second model, its role is limited to the interface between the agents and the user. The optimisation process, in this case, is distributed among the Resource agents.

Each agent in these models has its own acquaintances (i.e. the agents that it knows and with which it can communicate), a local memory composed of its static and dynamic knowledge and a mailbox in which it stores the messages received from the other agents. In the remaining of this section, we describe briefly each agent class in both models.

4.1 Centralized Optimisation Model

4.1.1 Job Agent

The acquaintances of Job agent are composed of Resource agents that are likely to fulfil its operations and of the Interface agent. Its static knowledge includes its release and due dates, its process routing and the different processing times of its operations according to the resources. Whereas its dynamic knowledge comprises the start times of its operations and the current resources to which they are allocated.

The Job agent is satisfied when all its operations have been affected to potential resources and when all its constraints are not violated and in this case it does nothing. Otherwise, it is unsatisfied and it tries

to assign an operation to an eligible resource in cooperation with its acquaintances.

4.1.2 Resource Agent

The acquaintances of Resource agent are composed of all Job agents whose operations are likely to be fulfilled by it and of the Interface agent. Its static knowledge encloses the list of operations that it can perform along with their processing times. While its dynamic knowledge is composed of the operations currently assigned to it and their start times.

The Resource agent is satisfied when no overlapping conflict exists between two operations assigned to it and in this case it does nothing. If not, it is unsatisfied and it tries to solve these conflicts by sending one of the conflicting operation to its Job agent in order to replace it elsewhere.

4.1.3 Interface Agent

The Interface agent acquaintances are composed of all the agents existing in the system. Its static knowledge contains:

- The maximal number of iterations allowed
- The tabu list size

Its dynamic knowledge is composed of:

- The tabu list
- The current solution and its makespan
- The best solution encountered so far and its makespan
- The current number of iterations performed.

As mentioned before, the Interface agent, in this model, contains the core of the optimisation process. The Interface agent remains unsatisfied until the current number of iterations exceeds the maximal number of iterations allowed. Otherwise, it delivers the best solution to the user.

4.2 Distributed Optimisation Model

4.2.1 Job Agent

The acquaintances of Job agent are composed of all Resource agents and the Interface agent. Its static knowledge includes its release and due dates, its process routing and the different processing times of its operations according to the resources. Whereas it has no dynamic knowledge.

The Job agent is satisfied when all its operations have been assigned, and in this case it does nothing. Otherwise, it is unsatisfied and it tries to assign an operation to an eligible resource in cooperation with its acquaintances.

4.2.2 Resource Agent

A Resource agent can communicate with all the other agents existing in the system. Its static knowledge encloses the entirety of the information concerning the operations; i.e. their processing times, their potential resources, their predecessors and successors according to their job process routings, etc.; along with the maximal number of iterations allowed, the number of iterations allowed between two successive improvements and its own tabu list size. While its dynamic knowledge is composed of its tabu list, its current solution and its cost, the best solution that it has encountered so far and its cost, the number of iterations that it has performed, the number of iterations since the last improvement made and a list of the best solutions reached by the other Resource agents.

The Resource agent is unsatisfied while the number of iterations that it has performed is less than or equal to the maximal number of iterations allowed. Otherwise, it is satisfied and it does not anything.

4.2.3 Interface Agent

The Interface agent acquaintances are composed of all the agents existing in the system. This agent has no static knowledge. However, its dynamic knowledge is composed of the list of the best solutions encountered by the Resource agents and the global best solution and its cost.

It is satisfied when all the other agents are satisfied and in this case it delivers the best solution found to the user. Otherwise, it is unsatisfied and it does nothing.

In the remaining of this paper, we present the Multi-Agent global dynamic in the two cases of initial solution and optimal solution determination.

5 MULTI-AGENT GLOBAL DYNAMIC

In this section, we describe the global dynamic of both Multi-Agent systems proposed for the FJSP. Two main phases compose this global dynamic: initial solution determination phase and optimisation phase based on TS. The former is similar in both models, whereas the latter differs from the centralized version to the decentralized one. The following section describes the initial solution generation process used for generating the initial solution from which the optimisation starts. Next, the optimisation processes of both models will be described.

5.1 Initial solution determination phase

The initial solution is the result of agent cooperation. Initially, the Interface agent creates the different Job and Resource agents and sends the message "Determine_Initial_Allocation(Jk)" to Job agents in order to find an initial allocation for all their operations. The job agent selects, consequently, the less loaded resource among the potential resources and a start time d such thatFor the first operation of a job (according to the process routing) d is equal to the release date of the job. Otherwise, d is equal to the finish time of its precedent operation.

Such an initial allocation satisfies all precedence and temporal constraints. However, it remains to verify the disjunctive constraints. Each time an operation is assigned to a resource, its Job agent informs the concerned Resource agent through the message *"Operation_allocated(R_l, O_i, d)"*. At the receipt of this message, the Resource agent R_l checks its satisfaction. In the case it is unsatisfied, i.e. there is an overlapping conflict between this operation and another operation that has been already affected to it, it tries to find another satisfying location on it which start time d_l is the closest possible to *d*. If such a location exists, then it informs the Job agent through the message *"Operation_modified(J_k, O_i, d_l)"*. Otherwise, it ejects the operation and sends it to its Job agent in order to search for another location through the message *"Operation_refused(J_k, O_i)"*. At this moment, the Job agent sends the operation to another potential resource.

The process above-mentioned will be repeated as many times as the operation is not yet assigned and for a predefined number of iterations. Once this threshold is exceeded, namely the Job agent has not found any location on a potential resource, it will request one of the possible resources to create a location through the message *"Create_location(R_x, O_i)"*. Such a location must satisfy all problem constraints. Similarly, if the Resource agent fails in creating such a location, it ejects the operation and sends it to its Job agent to contact another Resource agent, and so on. This process stops when a second predefined threshold has been exceeded.

5.2 Optimisation process

5.2.1 Centralized Optimisation Process

As mentioned before, in this first approach the core of the TS is implanted on the Interface agent. The latter generates the neighbourhood of the current solution and then chooses the best non-tabu move contained in it using its evaluation mechanism. This best move is then sent to the other agents in order to generate the feasible solution obtained by applying this move on the current solution, and so on.

In the following, we describe the parameters of the TS used in this model.

Neighbourhood function

A tabu search-based approach complexity depends essentially on (1) the current solution neighbourhood size and on (2) the evaluation scheme of this neighbourhood with which the best solution will be determined. It seems then interesting to reduce the size of the neighbourhood in order to reduce problem complexity.

To present the neighbourhood function of the centralized optimisation process, we need first define the notion of *critical path*. A critical path of a solution is the path which length is equal to the schedule one and that is composed of operations related to by either a precedence constraint, or a disjunctive constraint.

A *critical operation* is an operation which belongs to a critical path. The neighbourhood of a solution is obtained by two types of moves: Switch of two adjacent critical operations achieved by the same resource or migration of a critical operation on another potential resource.

Neighbourhood evaluation

The best non-tabu neighbour belonging to the current solution neighbourhood will be selected for the next iteration. Hence, all neighbours must be evaluated in order to determine the best one. However, a global evaluation, i.e. computation of all start times of all operations, of each neighbour will need a considerable time. For this reason, only a subset of operations will be taken into account and to which start times will be redefined. These operations are effectively concerned by the move executed.

Optimisation process

At the end of the first phase detailed earlier, the Interface agent receives the initial solution and launches then the second phase based on TS. The following algorithm presents the core of the optimisation process implanted in the Interface agent.

```
1.  tabu_list ← ∅
2.  nb_iter ← 0
3.  current_sol ← initial_solution
4.  best_sol ← current_sol
5.  While nb_iter <= nb_iter_max do
6.     iter_diversif ← 1
7.        While iter_diversif<=iter_max_diversif &
          nb_iter <=  nb_iter_max do
8.           path ← critical_path (current_sol)
9.           neighbH ← determine_ neighbH(path)
```

```
10.        best_N←determine_best_N(neighbH)
11.        tabu_list←add_in_tabu_list (best_N)
12.        current_sol ←perform_move
                        (current_sol,best_N)
13.        if cost(current_sol)<cost(best_sol) then
14.           best_sol ← current_sol
15.           iter_diversif ← 0
           End if
16.        nb_iter ← nb_iter +1
17.        iter_diversif ← iter_diversif +1
      End while
18.   Diversification
   End while
```

Once the best neighbour among the neighbourhood of the current solution has been chosen, the Interface agent sends the operation concerned by the move chosen to its Job agent in order to inform it about the move to perform. At the receipt of this operation, the Job agent sends it to the Resource agent R_l in order to find a location starting at a date d. The date d is equal to the finish time of the predecessor of O_i. The same process involving the cooperation between the agents, described in the first phase, will then be repeated in the case that this assignment leads to a conflict on resource R_l, otherwise, no changes are made.

When the number of iterations between two best solutions exceeds a predefined threshold "*iter_max_diversif*", a diversification phase is performed. The latter consists in varying the search in order to explore new regions of the search space. In our approach, such a phase is characterized by replacing some operations selected randomly. An operation is replaced on one of its potential resources selected also randomly.

5.2.2 Decentralized Optimisation Process

In this approach, we have distributed the optimisation process among the Resource agents. Each Resource agent will, from now on, have its own optimisation process and its individual parameters of its TS. The Resource agents will send mutually the best solutions encountered in order to help each other to diversify their search process.

In the following, we describe the parameters of the TS used in this model.

Neighbourhood function

The *neighbourhood* of a current solution in a Resource agent is obtained only by switching two operations achieved by this resource or by transferring an operation currently affected on this resource to another potential resource.

Neighbourhood evaluation

The best non-tabu neighbour belonging to the current solution neighbourhood will be selected for the next iteration. The subset of operations that will be concerned and to which start times will be recomputed is the same as defined for the first version.

Optimisation process

Once the initial solution has been determined, the Interface agent sends it to each Resource agent in order to start its local optimisation process. The Resource agent determines then, the neighbourhood of the current solution. After evaluating the neighbourhood, the Resource agent chooses the best non-tabu neighbour (*best_N*). When no non-tabu neighbour exists, the Resource agent continues its process from a solution already sent by another Resource agent and which has been stored in the list of best solutions (*list_best_solutions*). In the case that the latter is empty, a diversification phase will take place. When the neighbour is chosen, the move will be accomplished and the new solution will be obtained. In the case that the new current solution improves the best solution (*best_sol*) encountered so far, the Resource agent sends a message to the other Resource agents in order to add this solution to their lists of best solutions. When the number of iterations between two best solutions exceeds a predefined threshold "*iter_max_diversif*", a diversification phase is performed. The following algorithm presents the core of the optimisation process implanted in the Resource agent.

```
1.  tabu_list ← ∅
2.  nb_iter ← 1
3.  current_sol ← initial_sol
4.  best_sol ← current_sol
5.  While nb_iter <= nb_iter_max do
6.     iter_diversif ← 1
7.     While iter_diversif <= iter_max_diversif
       & nb_iter <= nb_iter_max do
8.     neighbH← determine_neighbH(current_sol)
9.     best_N← determine_best_N(neighbH)
10.    While best_N = nil
       & list_best_sol is not empty do
11.       alea←random_selection(list_best_sol)
12.       neighbH← neighbH ∪{alea}
13.       tabu_list ← ∅
       End while
14.    If best_N <> nil then
15.       tabu_list←add_in_list_tabu (best_N)
16.       current_sol ← perform_move
                        (current_sol, best_N)
17.       If cost(current_sol) < cost (best_sol) then
18.          best_sol ← current_sol
19.          iter_diversif ← 1
20.          loop on Resource agents Ri
21.             send_best_sol(Ri,best_sol)
             End loop
          End if
          Else
```

22. current_sol← Diversification(current_sol)
23. iter_diversif ← 1
24. tabu_list ← ∅
End if
25. nb_iter ← nb_iter+1
26. iter_diversif ← iter_diversif+1
End while
27. current_sol ← Diversification(current_sol)
28. iter_diversif ← 1
29. tabu_list ← ∅
End while

6 EXPERIMENTATION

Some experiments have been made on various benchmarks defined by (Chambers & Barnes, 1996), (Dauzerre & Paulli, 1997), etc. These benchmarks have a number of jobs varying in the set {10, 15, 20}, the number of resources in the range [5, 20], the number of operations per job in the range [5, 25] and the number of potential resources per operation in the range [1,3]. Consequently, the benchmarks considered have a total number of operations ranging in [50, 500].
The parameters used in the TS of each Resource agent are the following:

- Tabu list size varying in {8,10,15,20,30} for the centralized process and in {8,10} for the distributed one.
- Total number of iterations *nb_iter_max* fixed to 1000 in the centralized approach and to 300 in the distributed one.
- Number of iterations between two diversification phases varying in {250,300,350} for the centralized process and in {20,40} for the distributed one.

Table 1 presents the results obtained by both approaches for some instances among the benchmarks mentioned earlier as same as the lower and the upper bounds (*LB* and *UB*) presented in the literature for the same instances.

Table 1: Results for Lawrence & al. instances

Benchmark	*LB*	*UB*	*Centralized process*	*Distributed process*
la01	609	609	620	662
la02	655	655	666	704
la03	550	554	575	596
la04	568	568	581	675
la05	503	503	503	541

7 CONCLUSION

In this article, we have presented two Multi-Agent approaches for solving the FJSP. These approaches are based on the TS. The Multi-Agent systems proposed are composed of three agent classes: Job agents, Resource agents and an Interface agent. In the first approach, the core of the TS is implanted in the Interface agent and the other agents cooperate in order to generate a feasible solution from the best neighbour of the current solution. In the second approach, each Resource agent has its own TS and the Resource agents send mutually their best solutions encountered. Some experiments have been made on a plenty of benchmarks. The results in both approaches show that the solution provided is close to a range defined by the lower and the upper bounds given in the literature.

REFERENCES

Brucker, P. & Neyer, J. , 1998. Tabu-search for the multi-mode job-shop problem. *OR Spektrum 20,* 21-28.

Chambers, J.B. and Barnes, J.W., 1996. Flexible Job Shop scheduling by tabu search. Graduate program in Operations Research and Industrial Engineering, The university of Texas at Austin, Technical Report series, ORP96-09.

Dauzerre-Peres, S. and Paulli, J., 1997. An integrated approach for modeling and solving the general multi-processor job shop scheduling problem using tabu search. Annals of Operations Research 70: 281-306.

Glover F., 1986. Future paths for Integer Programming and Links to Artificial Intelligence,.*Computers and Operations Research*, 5:533-549.

Mastrolilli M., Gambardella L.M., 2000. Effective Neighborhood Functions for the Flexible Job Shop Problem. *Journal of Scheduling, Volume 3, Issue 1*. Pages:3-20.

Nuijten W., Aarts E., 1996. A computational study of Constraint Satisfaction for multiple capacitated Job Shop scheduling. *European Journal of Operations Research, 90(2)*: 269-284.