11th International Conference on Modeling, Optimization and Simulation - MOSIM'16
August 22-24
Montréal, Québec, Canada
"Innovation in Technology for performant Systems"

EMPIRICAL IMPROVEMENTS OF A DYNAMIC SCHEDULING ENGINE IN AN INDUSTRIAL ENVIRONMENT

Mahmoud MOHANNA

Département de génie de la production automatisée
École de la technologie supérieure, Montréal, Canada
mahmoud.mohanna.1@ens.etsmtl.ca

Amin CHAABANE

Département de génie de la production automatisée
École de la technologie supérieure, Montréal, Canada
Amin.Chaabane@etsmtl.ca

ABSTRACT: *This paper introduces the results of a series of empirical improvements performed on a basic algorithm addressing the Flow Shop Scheduling Problem in an industrial environment. Metaheuristic methods are followed to enhance the time and the quality of the initial solution produced by the scheduling engine of an industrial platform in order to obtain a rather acceptable initial schedule. Thereafter, several boosts have been achieved in order to accelerate the convergence towards an optimum solution besides the reduction of processing time and memory allocation. Further research work is required to improve resource assignation by making it more efficient and reasonable and to optimize memory allocation so that the current scheduling engine becomes more scalable and updatable.*

KEYWORDS: *Flow shop scheduling, Machine selection, Operation assignment, Heuristics, Metaheuristics.*

1 INTRODUCTION

In the majority of industrial environments, the classical approaches for scheduling are rarely able to take into account the new arising information which is more and more present in different information systems of enterprises. Thereby, the evolution from static approaches to dynamics ones allow to enhance the performance of the scheduling engine towards the quality of the proposed solution. Within this context, the continuous planning in order to find a new optimal solution according to the new arising information is always necessary. The utilization of a metaheuristic approach to solve the problem of dynamic scheduling is privileged in this project. The approach of dynamic scheduling allows not only the platform under study to maintain the confidence of existing clients and to satisfy their needs, but also to attract new clients whose hazards are frequent.

The rest of this paper is organized as follows: section 2 provides an overview of related work and a brief literature review about previous contributions in similar problems. Section 3 explains the research context and presents the problems to be addressed. In the next four sections; sections 4,5, 6 and 7, we illustrate the enhancements achieved so far on the existing platform. Afterwards, in section 8, we discuss further project work to be achieved as well as its directives. Finally, we present a brief conclusion summarizing the key elements of the paper contents.

2 LITERATURE REVIEW

Each manufacturer performs several tasks in order to satisfy growing client requirements (In the rest of this paper a task will be always referred as a job). Each job is composed of one or several operations. Each operation must be assigned to one machine. Thus, each job must be assigned to one or more resources (machines) in order to be completed. Accordingly, scheduling problems deal with the allocation of the available resources to required jobs over time. In most cases, resources used in manufacturing activities are limited. Thereby, scheduling becomes a very important aspect in managerial decision-making. This importance draws the attention of both practitioners and academicians to scheduling.

2.1 Open Shop, Job Shop, and Flow Shop Scheduling

Scheduling problems usually lie in the NP-hard problem class. The problem hardness increases considerably when the increase in number of jobs and/or in machines involved increases as well as the increase of job constraints and the interdependence within the jobs and/or between them. In the literature, three types of scheduling problems are identified; Open Shop Scheduling Problems (OSSP), Job Shop Scheduling Problems (JSSP), and Flow Shop Scheduling Problems (FSSP). In OSSP, there is no ordering constraint neither for the operations within the same job nor for the set of jobs on the floor. Moreover, JSSP are rather more complicated where operations and totally ordered within each job, but still no order constraint for the job set. Going more complicated, in FSSP, both operations and jobs must be ordered. Each

type of scheduling problems can be seen as a subset of its predecessor (Werner, 2011). It is evident that FSSP is the most complicated category due to the two dimensional ordering constraints; for operations within each jobs as well as for the entire set of jobs.

2.2 Dynamic Scheduling

In industrial environments, it is very often that job planning task is not done only once before starting the fabrication process. Nevertheless, a schedule produced by a scheduling engine is always subject to revision, modification, or even addition of more jobs during or after the termination of the scheduling process (Georgiadis and Michaloudis, 2012) and (Framinan et. al., 2014). Consequently, the need to implement a 'dynamic' scheduling engine arises (Ouelhadj and Petrovic, 2009). For this, classical scheduling approaches are no longer able to keep the scheduling engine update-to-date with the new arising information. It is evident that this evolution in the concept of scheduling is always accompanied by a superimposed complexity on the scheduling problem (Gao et. al., 2014), (Gen and Lin, 2014, 2014), and (Zhang et. al., 2012). Dynamic scheduling is indispensable in the presence of certain events in real time such as a new urgent arising job, a job cancellation, changes in due dates, priority changes, and/or changes in runtime. In this context, two sub-problems arise; the first is related to the identification of the moment when the new updates should be applied and the second one is whether to 'repair' this existing schedule or to reconstruct it entirely (Jin et. al., 2014). In the first case, it is no more than certain adjustments to be applied on the current schedule. However, in the second case, a new plan has to be produced. This necessitates an additional effort to produce a new optimum or near optimum solution within a reasonable processing time frame. Thus, the need to a 'smart' scheduling algorithm arises. Over the years, researchers in the combinatorial optimization community have tried to apply several metaheuristic algorithms in order to reach an optimum or quasi optimum solution for dynamic scheduling problems. Well-known local search algorithms such as the Genetic Algorithm at its derivatives (Davis, 1985), (Bierwirth, 1994), and (Kopfer et. al., 1993), Simulated Annealing (Kirkpatrick et.al.,1982), (Davis, 1987), and (Rojas, 1993), Threshold Accepting (Dueck and Scheuer, 1990), Tabu Search (Porto and Ribeiro, 1995), Flood Method (Dueck et. al., 1993), and Neural Networks (Williams and Zipser, 1989) and (Khaw, 1995).

The common idea between those algorithms is to produce new solutions starting from an initial solution and its successors or 'neighbourhoods'. The produced solutions are stored in the memory and used to produce further solutions until reaching a stop condition (Vaessens et el, 1996).

3 CONTEXT AND PROBLEM DESCRIPTION

The objective of the scheduling engine of the industrial platform under study is to generate an optimum, or at least acceptable, schedule for a large number of interdependent fabrication jobs in an industrial environment within a reasonable time frame. It is evident that the problems addressed by such scheduling engine lie in FSSP which is the most complicated type of scheduling problems where operations within the same job as well as the whole job set are order restricted.

The initial design of the employed scheduling engine is too elementary and greedy. The basic idea of the employed algorithm is, for a particular operation, to find the best resource as well as its best order within the chain of operations assigned to such resource. In the rest of this paper, we call such order as the index. Suppose that we have n operations assigned to a certain resource (machine). The first operation of them has the index no.1, the index of the second operation is 2, and the index of the n^{th} operation is n.

The current employed algorithm is fully deterministic. It is based on calculating the cost (in most cases, the cost is evaluated as the total job tardiness) for all possible solutions, then keeping only the best one and getting rid of the previous solutions. This algorithm is a typical example of *Brute Force Algorithms* which are based on the idea of examining all candidate solutions in order to elect the best one.

Suppose that we have n jobs (starting from J_1 to J_n) to be assigned to m machines (starting from M_1 to M_m). Each job is composed of a number of p operations from O_1 up to O_p. The employed scheduling algorithm is composed of two main steps:

3.1 Insertion

The objective of the first step of the scheduling process is to generate an initial schedule (we here call a schedule as a solution). For achieving that, the jobs are inserted in a sequential order on the available machines in a forward scheduling schematic. At the end of this process, an initial solution will be ready. However, it is often too far from being optimum or even near optimum.

3.2 Descent

'Descending' an operation means rescheduling it to start earlier than its current starting date. In other word, pushing back an operation over a particular resource. The objective of this step is to fulfil – whenever possible – the time gaps resulting from the sequential insertion that has taken place in the previous step.

In this step, a sample containing 1% to 10% of the whole set of operations is selected arbitrarily. The exact proportion is specified randomly at each iteration. Each of

those selected operations are 'descended' on *all* indexes of *all* compatible resources. This means that the operation at the position 'n' will be displaced in the position n-1 then n-2, and so on.
At every descent, if the obtained solution is a valid one, its cost is evaluated. If it is better the current one, the system user is notified about the existence of a new potential solution. Figure 1 shows the pseudocode of the algorithm used to carry out this step.

Sequential single step descent pseudocode

```
WHILE (True)
  Select a random number s between 1 and 10
  Select a random set P of operations represents s% of the whole set of operations
  FOR EACH p in P
      Get the list (R) represents the compatible resources of p
      FOR EACH r in R
          i = last index of r
          FOR EACH (i from n to 1) where n is the number of assigned operations to r
              Place operation at (i)
              IF obtained solution is valid THEN
                  Evaluate the cost
                  IF obtained cost is less than the least obtained cost THEN
                      least cost = current cost
                      better index = i
                  ELSE
                      CONTINUE
                  END IF
              END IF
          END FOR
          IF cost is less than least cost
          least cost = current cost
          better resource = r
      END FOR
      IF least cost < cost of obtained solution THEN
          solution cost = least cost
          Notify the user about the existence of a new solution
      END IF
  END FOR
END WHILE
```

Figure 1: Pseudcode of descente algorithm

This operation is of order O(n*p*m) where n is the number of jobs, p of operations in each job, and m is the number of available resources.

For a job set composed of less than 200 jobs, each is composed of 10 operations in average, this algorithm performs ideally. However, the performance degrades exponentially with the increase of the number of jobs. The current scheduling engine fails to generate an initial schedule for a job set composed of more than 1000 jobs.

3.3 Scheduling objectives

There are five available objectives which are inspired from the most common job sequencing priority rules (Chase et el, 2001). The cost of each obtained solution is evaluated according to one or more of these objectives. The client has the ability to precise the scheduler objective(s) and to fix their priorities. The values of evaluated costs are stored in the form of array whose top element corresponds the highest priority objective, the second element corresponds to the second priority objective, and so on. In order to compare two solutions, the two top elements of the cost array of each one are compared together and the solution having the lower value is considered the better one. If the values of the two top elements are equal, the next two elements are compared, and so on. The five objectives are:

- **Minimize Late Job Cost (tardiness):** This objective aims to minimize the total tardiness cost of scheduled jobs.
- **finish all jobs As Soon As Possible (ASAP):** This objective aims to finish all jobs ASAP, i.e. to minimize the latency time. The difference between the objective and the previous one is that here further optimization can take place even though all jobs are scheduled to be finished before their expected end dates.
- **Finish prioritized jobs first:** In this objective, some jobs are given higher priorities more than other. Such jobs of higher priorities must be finished first regardless their expected end date.
- **Minimize Just In Time cost (JIT):** This objective aims to minimize the whole production time for each job in order to reduce inventory costs. Tardiness and earliness are both involved in evaluation of this cost.
- **Minimize Delivery date priority cost:** This objective aims to prioritize jobs having closer due dates to be finished before other jobs.

The first two objectives, respectively, are the most used by clients. Thereby, simulation results are usually evaluated according to them. All simulation graphics illustrated in this paper are all generated according to the 'Minimum Late Job Cost' objective.

4 INITIAL INSERTION

The first problem to be fixed is to get a quick acceptable (to certain extent), if possible, initial schedule for a relatively large number of jobs.

In order to get an initial schedule, the current implemented algorithm loops over all compatible resources for each new added operation passing through the following steps:

1. Assign the new operation to the first resource in the list of compatible resources;
2. If succeeded[1], remove the operation from the current resource;
3. Assign it to the next resource in the list, then go to step 2;
4. At the end of the loop, assign the operation to the resource that implies the minimum solution cost;
5. Repeat all previous step for all operations for all jobs.

4.1 Improvement

In the insertion step (section 3.1), instead of examining all compatible resources for each operation to get the best resource which gives the minimum cost in order to

[1] The word 'succeeded' here means that the operation can be assigned to the selected resource at the new index without violating the precedency constraint, i.e. it does not proceed any other operation that must be finished before that it starts.

assign the current operation to it, the scheduling engine can select the resource having the earliest end date and consider it directly as the best resource to assign the current operation. That is not all. Furthermore, the new operation is inserted in the first fitting available time gap instead of adding it at the end of the chain of the selected resource (if applicable).

On the same scheduling server and under the same calculation and processing conditions: For a number of 897 jobs, each one is composed of 10 operations in average, the new procedure produces an initial schedule within a period of 3 minutes instead of 41 minutes using the old one.

5 SEQUENTIAL VARIABLE STEP DESCENT

This approach tends to accelerate the convergence towards an optimal solution as well as to escape from potential local optima. As explained earlier, in the decent stage in the current procedure, each operation of the descent sample is placed in each index of each compatible resource. Actually, it is 'descended' step by step over each of the compatible resources. This is repeated for each operation in the descent sample (between 1% to 10% of the total number of operations). Instead, in the proposed approach, the size of the backward step[2] becomes variable. The size of each single step is determined according to two factors:

1. The tendency of the cost curve, whether the cost of the produced solutions increases or decreases. In case of cost diminution, whether it converges or diverges.
2. The number of operations assigned to the same machine.

For the first factor, we differentiate the cost curve twice in order to get its first and second derivatives. The first derivative tells us if the cost tends to raise or to decline. If the cost curve declines, the second derivative informs us if it converges towards a local optimum or not. Thus, we have three possibilities:

a. If the cost increases, a significant increase will be applied to the backward step size in order to escape from the region where cost increases to another one. This helps also in escaping from local minima when the curve starts to rise after reaching one of them;
b. If the cost decreases, the backward step size will increase proportionally with the diminution of the cost;
c. Backward step size will vary according to the degree of change in the cost. If the cost decreases and converges, backward step size will be increased and vi-sa versa. This is done in order to accelerate the convergence towards a good solution in the first case, and to avoid missing a good solution in the second one. The next figure (figure 2) presents a pseudo-code of the sequential variable step algorithm.

Sequential variable step descent pseudocode

```
1   WHILE (True)
2     Select a random number s between 1 and 10
3     Select random set P of operations represents s% of the whole set of operations
4     FOR EACH p in P
5         Get the list (R) represents the compatible resources of p
6         FOR EACH r in R
7         i = last index of r
8         operation scalar = (number of assigned operations to r + x/2) mod x (increase step size
9                     for each x assigned operations)
10        backward step = 1 + operation scalar
11              WHILE (i > 0)
12                    Place operation at (i)
13                    IF obtained solution is valid THEN
14                        Evaluate the cost
15                        d1= evaluate first derivative of cost curve
16                        IF d1 < 0 THEN
17                            least cost = current cost
18                            better index = i
19                            d2 = evaluate second derivative of cost curve
20                            IF d2 > 0 THEN
21                                backward step = a rather big empirical value
22                            ELSE
23                                backward step = an intermediate empirical value (bigger than 1)
24                            END IF
25                        END IF
26                    END IF
27                    i = i - backward step
28              END WHILE
29              IF cost is less than least cost
30              least cost = current cost
31              better resource = r
33        END FOR
34        IF least cost < cost of obtained solution THEN
35            solution cost = least cost
36            Notify the user about the existence of a new solution
37        END IF
38    END FOR
39  END WHILE
```

Figure 2: Sequential variable step descente algorithm

In regard to the second factor mentioned above, this tends to take into consideration the number of operations assigned to a particular machine. More specifically, to increase the backwards step size in function of that number. For example, if more than 1000 operations are assigned to the same machine, it is too costly and slow to test assigning a single operation at each index of them. Instead, the backward step size will increase in a direct relation with the number of operation assigned on the same resource.

The choice of changes in step size is empirical. It depends on the absolute value of costs, the fluctuation degree as well as the total number of operations. For example, in our tests, 7 steps are added when cost curve rises, 3 steps if cost curve declines and converges, and one step in case of cost divergence. Furthermore, one additional step is for each 100 assigned operations.

Figure 3 and figure 4 (the X axis represents the number of valid obtained solutions whereas the Y axis represents the total job tardiness in milliseconds) show the test results on test database and real client database. The two figures compare the performance of the scheduling engine when applying the current rule (single backward step) and when applying the variable step rule without and with taking into consideration the number of machine assigned operations respectively. Figure 3 shows that for a real industrial database, only five solutions are obtained in the first case versus 41 and 49 solutions in the second and third cases respectively for the same

[2] Backward step is the difference between the current index of a particular operation and its new index when displacing it towards the beginning of the list of operations assigned to a particular resource.

number of jobs (500 jobs) within the same processing time (35 minutes). The enhancement of the quality of the obtained solutions is remarkable.

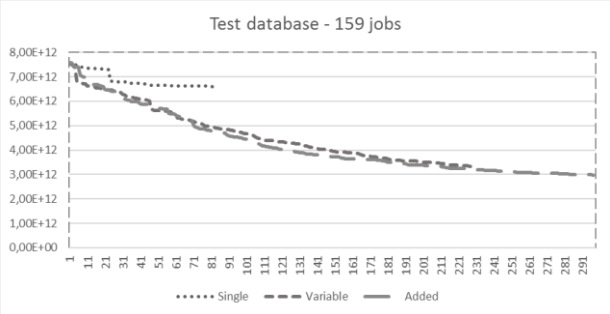

Figure 3: Comparison between three different approaches to determine backward step size on a test database

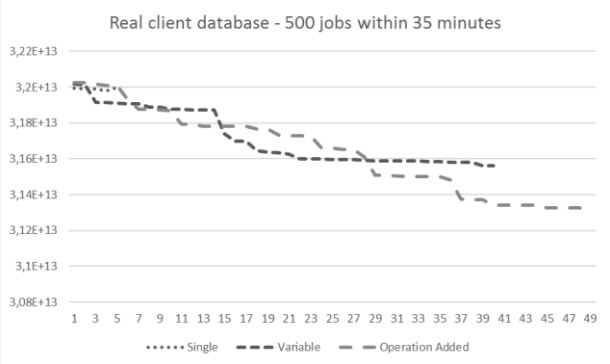

Figure 4: Comparison between three different approaches to determine backward step size on an industrial database

6 BINARY DESCENT

The last method results in a considerable improvement in the performance of the scheduling engine. Nevertheless, it is still of order O(p) on a single resource, where p is the number of operations assigned to such resource. Even with the variable step approach, the performance of the scheduling engine degrades significantly with the increase in number of jobs, and hence the number of operations to be scheduled.

The idea of the Binary Descent (pseudocode presented in figure 5) in is inspired from the Binary Search. Starting from the end of chain, the scheduling engine tends simply to place the selected operation in the middle of the operation chain of a particular resource. If the cost increases, the operation will be moved back and placed in the middle of the second half of the chain. Otherwise, the operation will be pushed towards the beginning of the chain and placed in the middle of the first half. This process is repeated iteratively until finding the best index to place the operation on the resource chain.

Binary descent pseudocode

```
1   WHILE (True)
2     Select a random number s between 1 and 10
3     Select a random set P of operations represents s% of the whole set of operations
4     FOR EACH p in P
5         Get the list (R) represents the compatible resources of p
6         FOR EACH r in R
7               i = last index of r
8               index limit = 0
9               WHILE (i > index limit)
10                    new index = (i + index limit +1)/2
11                    Place operation at (new index)
12                    IF obtained solution is valid THEN
13                        Evaluate the cost
14                        IF obtained cost is less than the least obtained cost THEN
15                            least cost = current cost
16                            better index = i
17                            i = new index
18                        ELSE
19                            index limit = new index
20                        END IF
21                    END IF
22              END WHILE
23              IF cost is less than least cost
24              least cost = current cost
25              better resource = r
26        END FOR
27        IF least cost < cost of obtained solution THEN
28            solution cost = least cost
29            Notify the user about the existence of a new solution
30        END IF
31    END FOR
32  END WHILE
```

Figure 5: Binary descente algorithm

Binary Descent reduces the time required to find the better index to place a selected operation to the order O(log p) instead of O(p). This results in a significant enhancement in the performance of the scheduling engine as shown in figure 6.

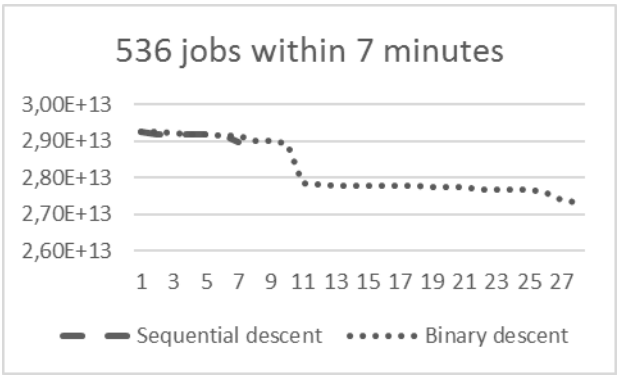

Figure 6: Comparison between sequential and binary descent methods

Figure 6 compares the performance of the scheduling engine for 536 jobs of a real industrial database within only 7 minutes of processing time. Only 7 solutions are obtained with the variable descent sequential approach while 28 solutions are obtained by applying the Binary Descent method. It is remarkable also that the cost curve declines significantly in the second case.

7 ARBITRARY CHOICE OF RESOURCES

During the descent, the scheduling engine determines the best resource for assigning a particular operation by executing an exhaustive search over all indexes over all resources as well. Despite of the fact that this method always returns the best resource to assign the selected operation in a deterministic way, it is too slow and exhaustive and as well as it degrades dramatically the system performance especially when looping over a large num-

ber of resources and/or when the number of operations is rather large. This process is of order of order O(p*m) where p is the number of operations assigned to a single resource and m is the number of resources (machines).

Alternatively, we opted to select a single arbitrary resource at each iteration among the list of compatible resources for a particular operation in order to assign it to (Random Selection Rule). Afterwards, we perform the binary descent over the arbitrarily selected resource in order to find the best index for such operation. We tried another approach tending to select the least charged resource and consider it as the best one instead of selecting it arbitrarily. However, that approach does not lead to satisfactory results as shown in figure 7.

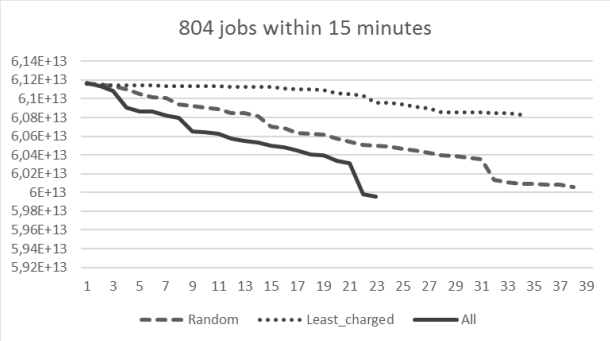

Figure 7: Comparison between three different approaches for best resource selection

The above graph compares between the three mentioned approaches for the same number of jobs and under the same computing conditions. From the figure we can notice that the arbitrary choice of a resources converges towards an optimum solution. For a large number of jobs, it is impossible to get a solution by using the exhaustive research method and thus the arbitrary selection of resources stays the most accepted approach so far.

7.1 Imperial results for scheduling 1054 jobs in a real industrial database

1. Looping over all resources: No results, system crashes!
2. Arbitrary selection: Four solutions obtained within 45 minutes under the same processing conditions as above.

For the scheduling engine under study, this is the first time ever to have a solution resulting from the iteration process (the descent step) after the generation of the initial schedule for such number of jobs.

8 FUTURE WORK DIRECTIVES

Despite of the fact that considerable improvements have been achieved, there are still a lot of work to be done in order to satisfy the rising client requirements. Future work directives could be summarized into the following three points:

8.1 Memory optimization

It is remarkable that memory consumption increases dramatically with the increase of number of jobs and/or the number of machines. For higher numbers of jobs (above 1000 jobs ≈ 10,000 operations) the current system usually crashes. Moreover, the current scheduling engine is not able to keep more than one solution in the memory. It always compares the new obtained solution with the current stored one. If the scheduling engine finds it better, it gets rid of the current solution and replaces it by the new one. This implies that all well-known scheduling algorithms presented in section 2.2 are not applicable in the current context. Memory optimization and improved utilization of system resources would open the door to implement and to apply smarter algorithms as well as the possibility to acquire more clients having higher number of fabrication jobs to be scheduled.

8.2 Smart selection of descended operations

As explained in the descent step in section 3.2, a random sample representing 1% to 10% of the total number of operations to be scheduled are selected to be 'descended' over the list of available resources. This sample is selected randomly without any precondition. Experiments are done in order to bias such selection by targeting only late operations or all operations of late jobs, but no satisfactory results are obtained so far. A smart selection of these operations might improve the quality of the obtained solutions through the descent process as well as accelerating the convergence towards an optimum solution.

8.3 Smart selection of resources

As shown in section 7, the most feasible way obtained so far for selecting a resource to assign a particular operation is to choose it randomly. Several experiments took place in order to improve this process by considering the least charged resource as the best one or by considering it which has the earliest due date (Earliest Due Date rule), but no satisfactory results are obtained so far. Further work should be done in order to find a smart satisfactory way for getting the best resource to assign a certain operation instead of choosing it arbitrarily. First-In-First-Out (FIFO) or Shortest Processing Time (SPT) rules might lead to better results in that regard (Subramaniam et. al., 2000).

9 CONCLUSION

In this paper, we presented an experimental research for the dynamic scheduling engine of an industrial platform. Considerable empirical improvements have been achieved in terms of the processing time and the quality of the initial schedule as well as subsequent scheduling solutions. These improvements are realized by following

a series of experimental steps; hypothesis, implementing, applying, testing, and comparing the obtained results with those obtained by the actual scheduling algorithm under the same scheduling conditions and the same number of jobs.

Next research and optimization work includes the optimization of memory usage in order to let the system to keep several scheduling solutions in order to be able to generate subsequent new neighbourhood-based solutions as well as the development of a new algorithm permitting the scheduling engine to select the resources smartly instead of performing this task arbitrarily or through an exhaustive search.

ACKNOWLEDGMENTS

This research project is funded by Mitacs Inc. through the accelerate program under grant no. IT06296.

REFERENCES

Chase, R.B., Aquilano, N.J. and Jacobs, F.R., 2001. Operations management for competitive advantage. *Irwin/McGraw-Hill.* Chapter 15.

Davis, L., 1985, July. Job shop scheduling with genetic algorithms. In *Proceedings of an international conference on genetic algorithms and their applications* (Vol. 140). Carnegie-Mellon University Pittsburgh, PA.

Davis, L., 1987. Genetic algorithms and simulated annealing.

Dueck, G. and Scheuer, T., 1990. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, *90*(1), pp.161-175.

Dueck, G., Scheuer, T. and Wallmeier, H.M., 1993. Toleranzschwelle und Sintflut: neue Ideen zur Optimierung. *Spektrum der Wissenschaft*, *3*(93), p.42.

Framinan, J.M., Leisten, R. and García, R.R., 2014. Overview of Manufacturing Scheduling. In Manufacturing Scheduling Systems (pp. 3-18). Springer London.

Gao, H., Kwong, S., Fan, B. and Wang, R., 2014. A hybrid particle-swarm tabu search algorithm for solving job shop scheduling problems. *Industrial Informatics, IEEE Transactions on*, *10*(4), pp.2044-2054.

Gen, M. and Lin, L., 2014. Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. *Journal of Intelligent Manufacturing*, *25*(5), pp.849-866.

Georgiadis, P. and Michaloudis, C., 2012. Real-time production planning and control system for job-shop manufacturing: A system dynamics analysis.*European Journal of Operational Research*, *216*(1), pp.94-104.

Jin, X., Liu, H.H., Gandhi, R., Kandula, S., Mahajan, R., Zhang, M., Rexford, J. and Wattenhofer, R., 2014, August. Dynamic scheduling of network updates. In *ACM SIGCOMM Computer Communication Review* (Vol. 44, No. 4, pp. 539-550). ACM.

Khaw, F.C., Singapore Computer Systems, 1995. *Neural network system and method for factory floor scheduling*. U.S. Patent 5,432,887.

Kirkpatrick, S., Gellatt, C.D. and Vecchi, M.P., 1982. Optimization by simulated annealing, IBM Thomas J. *Watson research Center, Yorktown Heights, NY*.

Kopfer, H. and Mattfeld, D., 1993. Genetische Algorithmen und das Problem der Maschinenbelegung. *Bremen, Germany*.

Ouelhadj, D. and Petrovic, S., 2009. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, *12*(4), pp.417-431

Porto, S.C. and Ribeiro, C.C., 1995. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints.*International Journal of high speed computing*, *7*(01), pp.45-71.

Subramaniam, V., Lee, G.K., Ramesh, T., Hong, G.S. and Wong, Y.S., 2000. Machine selection rules in a dynamic job shop. The International Journal of Advanced Manufacturing Technology, 16(12), pp.902-908.

Vaessens, R.J.M., Aarts, E.H. and Lenstra, J.K., 1996. Job shop scheduling by local search. *INFORMS Journal on Computing*, *8*(3), pp.302-317.

Werner, F., 2011. Genetic algorithms for shop scheduling problems: a survey.*Preprint*, *11*, p.31.