Journal of Advanced Research in Science and Technology
ISSN: 2352-9989

Tabu Search & Gpu-Based Genetic Algorithm to Solve The Job Shop Scheduling Problem With Blocking

A. AitZai [1*], A. Dabah [2] and M. Boudhar [3]
[1,3] USTHB / RECITS Laboratory, BP 32 Bab Ezzouar, Algiers 16111, Algeria.
[2] CERIST Research Center, Algiers, Algeria

Abstract.

This paper deals with the resolution of the job shop problem with blocking, where the machines have a limited or no storage space. To solve this problem, we compare between two different metaheuristics based on Tabu Search TS and Genetic Algorithms GA. The first one lays on a very efficient neighbourhood exploring and evaluation techniques, which improve the reliability of the method. These techniques operate on the critical path found in the alternative graph and always construct feasible solutions. The second lays on two different paradigms of parallelization with GA. The first uses a network computers and the second use GPU with CUDA technology. The results are very interesting. In both methods, we obtain a very significant reduction of computation time compared with the existing literature results.

Keywords: *Job shop, Blocking, Tabu search, Neighborhood, Genetic algorithm, GPU;*

1. Introduction

In the literature, the job shop scheduling problem (JSB) is defined as follows: a set of n jobs where each job i (i=1,…,n) is composed of ki operations, have to be processed on a subset of m specified machines (without going back to the same machine) according to a given order. Each machine has to process one operation at a time without interruption. Taking into account all these constraints, our aim is to find a schedule of all operations such that the latest completion time operation is minimized. The job shop problem with blocking (JSPB) is a version of the (JSB) with no-intermediate buffers, i.e. where machines have no storage space thus a job has to wait on a machine until it is processed on the next one.

Several papers have tackled the resolution of this problem but its NP-Hardness, described in detail by Hall and Sriskandarajah (1996), is the real obstacle to find an exact solution. Moreover, researchers have not succeeded in finding an exact method to solve the JSPB with more than 10 jobs × 10 machines. For this reason, they have turned their attention into the investigation of meta-heuristics which, though they do not guarantee to find the optimum, can in general reach good solutions in acceptable time.

Major investigations in the literature consider the flow shop problem with blocking (FSPB), while the number of papers that have tackled the JSPB is poor. Since the publication of the first work dealing with the JSP, a great effort has been made for the design of branch-and-bound (B&B) algorithms. Among others, we can cite Carlier and Pinson (1989) algorithm which solves the problem with size 10×10 proposed by Fisher and Thompson (1963).

* Corresponding author.
E-mail: h.aitzai@usthb.dz, h.aitzai@gmail.com (Ait Zai A)
Address: BP 32 Bab Ezzouar, Algiers 16111, Algeria

Then, Carlier and Pinson (1994) introduced the possibility of fixing the alternative arcs without branching, which made their algorithm very efficient. The majority of subsequent works were based on their results. Further efforts have focused on approximate methods and metaheuristics, Brizuela et al. (2001) have developed a Genetic algorithm for solving the job shop problem with blocking and no-wait constraints and they presented numerical results for four selected benchmarks. Klinkert (2001) and Gröflin and Klinkert (2004) introduced a generalized disjunctive graph for modeling different types of scheduling problems, in addition, they developed a local search method for the job shop problem with generalized blocking constrain. Gröflin and Klinkert (2009) proposed a new neighborhood function for the Tabu Search applied to the JSPB which always gives feasible neighbors. As for the parallel approach, a parallel implementation of the B&B method has been proposed by Perregaard and Clausen (1998), this algorithm is based on the results of Carlier and Pinson (1989) and Brucker et al. (1994). AitZai et al. (2012) have proposed a parallel genetic algorithm based on the master/slave model with a rule-based priorities coding and a parallel branch-and-Bound procedure for the JSPB.

Ben Mabrouk (2009) proposed a parallelization of a hybrid genetic–Tabu search-based algorithm for solving the graph coloring problem. Work on the parallelization of GA is scarce. Indeed, several parallelization techniques are proposed in the literature, especially for genetic algorithms, such as fine-grained methods (Gorges-Schleuter, 1989; Manderick and Spiessens, 1989; Mühlenbein, 1989) and the parallel hierarchical methods (Gruau, 1994), but works on the parallel GA using GPU are more rare.

The Tabu search metaheuristic has been applied to the JSPB by different authors, Mati et al. (2001) and Gröflin and Klinbert (2009). The classical neighborhood used in the literature, consists in replacing an alternative arc in the critical path by its alternative. This generally leads to new unfeasible neighbor solutions. To correct this, Mascis and Pacciarelli (2002) have proposed an algorithm that restores the solution's feasibility. But this technique increased the execution time.

Recently, Pranzo and Pacciarelli (2016) presented an iterated greedy metaheuristic, Dabah et al. (2016) described a parallel Branch and Bound method, Louaqad and Kamach (2016) elaborated a mixed Integer Linear Programs for Blocking and No-Wait Job Shop Scheduling Problems in Robotic cells and Dabah et al. (2017) proposes a new tabu search neighborhood based on reconstruction strategy. This neighborhood consists to remove arcs causing the infeasibility and rebuild the neighbor solutions by using heuristics.

In this paper, we try to contribute to these efforts by investigating, at first the performance of Tabu Search procedure in the resolution of the problem. Therefore we used a neighbourhood exploring and evaluation techniques proposed in Dabah et. al (2017), which accordingly improve the reliability of the method. These techniques operate on the critical path found in the alternative graph and always construct feasible solutions. In the second time, we use two different paradigms of parallelization with GA. The first ones use network computers Aitzai and Boudhar (2013) and the second use GPU with CUDA technology AitZai et. al (2013). In both methods we have obtained a very significant reduction of computation time, compared with the existing literature results. Our aim is to improve the results found in the literature.

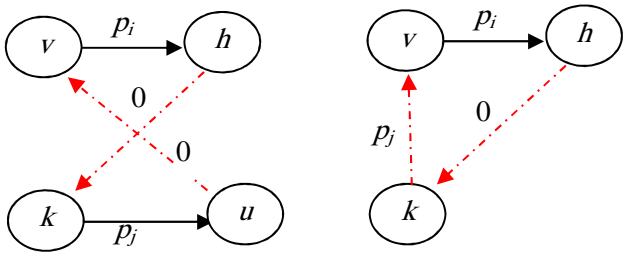
The remainder of this paper is organized as follows; Section 2 presents the problem formulation which is based on alternative graphs. Section 3 provides the Tabu search metaheuristic and some application techniques that enabled us to improve its performance. In section 4 we give the details of our parallelization methods based on CPU and GPU models. Comparison and analysis of the results obtained by the proposed methods is given in Section 5. Finally, Section 6 provides a general conclusion and gives some prospects for future work.

2. Problem Formulation

This section is concerned with the modelling of the JSPB using alternative graphs as detailed in Mascis and Pacciarelli (2002).

The simple job shop scheduling problem is usually modelled in two different ways: either by linear programming or conjunctive graphs. In the latter, the orientation of the conjunctive arcs defines the order in which the various operations are performed on any given machine. With the advent of the blocking constraint, the latter cannot be modelled by conjunctive graphs; this is why another modelling approach is used in graph theory, called modelling by alternative graphs. In this type of graphs, a new concept, alternative arcs, is introduced. Two operations connected by an alternative arc with weight zero, means that these operations can start at the same time, unlike disjunctive arcs that define a priority relationship between the operations. In this modelling approach, the vertices of the alternative graph represent the operations and the arcs defined are of two types: the conjunctive arcs (the priority arcs between the operations of the same job) and the alternative arcs.

Figure 1 Example of a blocking operation



The formulation by alternative graphs is a generalization of disjunctive graphs, Carlier and Pinson (1994). It is based on the triple (N, F, A) which represent the set of operations, the set of conjunctive arcs and the set of alternative pairs of arcs respectively. All arcs $(i,j)\in F$ are fixed initially and f_{ij} is the arc's weight. Each element $((u,v),(h,k))\in A$ is a pair of alternative arcs (u,v) and (h,k). Let $a_{uv}\geq 0$ and $a_{hk}\geq 0$ be the weights of the arcs (u,v) and (h,k) respectively. Let u and k be concurrent operations on the same machine j. We say that u or k is a blocking operation, because one of them must be processed first on j (see Figure 1).

3. Tabu Search technique

TS is a local search method. It consists in exploring the neighbourhood $N(s)$ of a current solution s. At each iteration the best solution $s'\in N(s)$ is chosen as the new solution, (even if its quality is worse than the current solution s). This strategy can lead to cycles, to avoid them we store the last k solutions in a short term memory (*stm*) called Tabu list. In the process of TS, all solutions in *stm* are prohibited. With the iterative computation process, it is possible to find an improved solution s belonging to the *Tabu list*. So, it is possible to accept this solution, despite its Tabu restriction, if it satisfies aspiration criteria.

The principle of Tabu Search is given by the following algorithm.

Algorithm TS;
Step 1. Find an initial solution s_0.
Step 2. Let $z=s_0$, i=0 and TL=Ø (TL: Tabu list)
Step 3. i=i+1;
Step 4. Find the neighborhood subset $N^*(s_i)\subseteq N(s_i)$ thus, $\forall\ s_{i+1}\in N^*(s_i) : (s_i,s_{i+1})\notin TL$;
Step 5. Replace s_i by s_{i+1} such that s_{i+1} is the best solution in $N^*(s_i)$;
Step 6. Update TL;
Step 7. If Stopping condition is not satisfied, Go to 3;

End.

3.1.Tabu Search Application

TS has shown its efficiency in solving many optimization problems. In this section, we present the implementation of:

[1] generating an initial solution,
[2] defining a local neighborhood search procedure,
[3] defining an evaluation function,
[4] defining the Tabu List and the Aspiration Criteria.

At first, we generate a random initial solution, using the priority rule-based encoding. This type of coding allows us to have every time a feasible solution thanks to the decoding algorithm.

The efficiency of this method depends mainly on the neighborhood function. Therefore, we present in the sequel, a new technique which improves the neighborhood function given by Gröflin and Klinkert (2009). To do this, we need the representation based on alternative graphs, presented in Section 2, and some definitions given below (Mascis and Pacciarelli, 2002).

Definition1: A selection S is a set of alternative arcs obtained from A, where only one element per pair is selected. Thus, for all $((u,v),(h,k))\in A$, if $(u,v)\in S$ then $(h,k))\notin S$.

Definition2: We say that a selection S is complete if one arc is selected from each pair of alternative arcs in the set A.

Definition3: Let $((u,v),(h,k))\in A$ a pair of alternative arcs. We say that the arc (u,v) is selected in S if $(u,v)\in S$. Otherwise, (u,v) is excluded from S if $(h,k)\in S$. Also we say that the pair is unselected if neither $(u,v)\notin S$ nor $(h,k)\notin S$.

Definition4: For any selection S, let $G(S)=(N,F\cup S)$. S is consistent if $G(S)$ has no positive length cycles.

Definition5: Let (i,j) be an alternative arc in $G(S)$, we call $tail(i,j)$ the job containing operation j, $head(i,j)$ the job containing operation i and $l(i,j)$ the longest path value from i to j.

Using this notation, any solution (schedule) is associated with a complete consistent selection S in the alternative graph $G(S)$. Its evaluation $V(G(S))=l(0,z)$ is the value of the longest path P in $G(S)$; where 0 and z are the *Source* and the *Sink* of $G(S)$ respectively.

Let a complete consistent selection S and $(u,v)\in P$ be given. The transition from S to a new complete selection S' is performed by interchanging an arc $(u,v)\in P$ (in the critical path) with its alternative (h,k), where $((u,v),(h,k))\in A$.

In most cases, this neighborhood function gives unfeasible solutions; therefore Gröflin and Klinkert (2009) have proposed an algorithm to fix unfeasible solutions. In this section, we describe a significant improvement in this neighborhood function that we shall detail later.

Let T and S be a set of alternative arcs and a complete consistent selection respectively; $T=\emptyset$. The construction of S' from S is done as follows:

[1] Choose an arc (i,j) from S and remove it ($S=S-\{(i,j)\}$), and set $T=T\cup\{(i,j),(i_a,j_a)\}$ such that (i_a,j_a) is the alternative arc of (i,j) in A.
[5] Remove all arcs (p,q) in S ($(S=S-\{(p,q)\})$) that verify : $tail(p,q)=tail(i,j)$, and set $T=T\cup\{(p,q),(p_a,q_a)\}$, such that (p_a,q_a) is the alternative arc of (p,q) in A.
[6] We use the AMCC Algorithm presented in Mascis and Pacciarelli (2002) to complete in S' the removed arcs from S. AMCC takes as input the graph $G=(N,F\cup S)$, then completes the arcs that were removed in steps (1) and (2) by choosing from T the pair of arcs (i,j) , $(h,k)\in T$ which satisfy the condition:

$l(0,h)+a_{hk}+l(k,z)=\max\{l(0,u)+a_{uv}+l(v,z)\}$; $\forall(u,v)\in T$; where 0 and z are the *Source* and the *Sink* of $G(S)$ respectively; and a_{hk} is the weight of the arc (h,k). This means that we choose the arc (h,k) that maximizes the value of *Cmax*.

Algorithm *Neighborhood function*
Input: *G= (N, F∪S), T≠Ø and S'=Ø.*
Output: *complete consistent selection S'.*
while *(T≠Ø)*
do begin
1. Select the alternative arc pair ((h,k),(i,j))∈T by ***AMCC;*** *let (h,k) be the arc to be added to S':*
2. S'=S' ∪{(h,k)}, T:=T- {((i,j),(h,k))};
3. ***while*** *(∃((u,v),(p,q))∈T, such that l(v,u)+a_{uv} >0)*
do begin *S'=S'∪{(p,q)};*
T=T−{((u,v),(p,q))} ;
end;
end;
End.

By the algorithm above, we construct feasible and appropriate neighboring solutions.

3.2.Solutions' Evaluation

The efficiency of a local search approach depends heavily on the evaluation method. However, each solution's evaluation, in our case, must compute the longest path in *G(S)* and this takes considerable time. It has been shown that nearly 90% of the total execution time is performed by the evaluation of solutions (Gröflin and Klinkert, 2009). Taking into account this consideration, we have implemented an evaluation strategy that calculates only the starting times of a subset of operations which is being involved by our neighborhood function algorithm given above. This is done, without repeating the whole longest path calculation.

3.3.Short and long Term List

Such neighborhood function may lead easily to recently visited solutions which form a cycle. This means that failing to find a better solution among them, does not guarantee that no better solution exists. This is why the best solution found by such Neighborhood function is called local optimum. To improve efficiency of this kind of neighborhood function we have used a Tabu List which avoids solutions leading to cycles. In order to do so, we put in a FIFO Tabu List, the *k* last solutions. It's not interesting to store all the elements composing the solutions in the Tabu List; this may generate a cumbersome list. So *k* is the size of the Tabu List and its elements must include sufficient information which accurately reflects the visited solutions. In our case, the Tabu list is composed, only, of the numbers of the alternative pair arcs affected by the transformation.

To avoid rotating around solutions with cycle sizes greater than *k* (*k* is the size of the Tabu List), we use the Long-term memory for storing values of *Cmax* already visited. A check of each identical *Cmax*'s sequences is done at every neighborhood function end.

3.4.Aspiration Criteria

We have taken over the aspiration criteria adopted by several applications of Tabu Search to Job Shop problem, this criteria is to remove the Tabu status of any solution when it leads to a better one.

4. Parallel Genetic Algorithm

In this section, we describe two different approaches of priority rules-based encoding GA parallelization. The first one is based on CPU networking and the second is based on GPU.

The basic design of metaheuristics offers improved strategies for finding solutions to combinatorial optimization problems. Parallelism offers an improvement of these methods by intelligently exploring many more solutions in very satisfactory times. By applying convenient parameters, parallel metaheuristics are more robust than their sequential versions in terms of the quality of the solutions obtained.

When solving the job shop scheduling problem with blocking in an exact way (AitZai et. al 2012), we noticed very large number of solutions explored to reach the optimal solution. However, a lot of works are oriented towards heuristics to solve the problems of sizes bigger than 10 jobs × 10 machines. Despite this, the quality of the solutions returned by these approximate methods still requires improvements through integrating parallelization techniques.

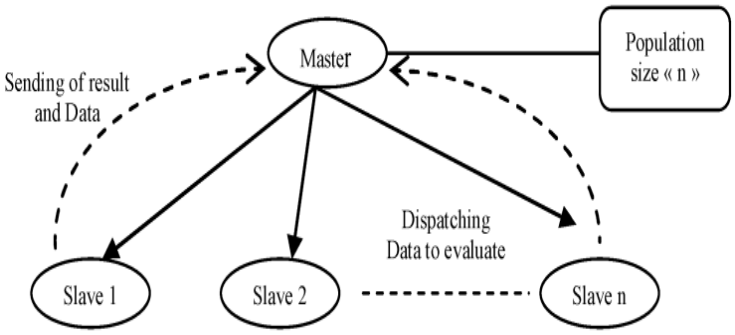
Thanks to their structure, GA adapt very well to parallel execution. So, with the growing popularity of parallel computers, parallel versions of GAs have been introduced and are subject to much research.

4.1.Using CPU

The parallelization of the GA can be achieved in two different ways: The first way uses the principle of multiple communication populations and the second one uses the master-slave principle. Both types of parallel GA have been widely used to reduce the execution time of many applications. The choice between the two parallel methods is determined by several factors, such as ease of use or execution and their potential to reduce execution times. Generally, parallel GA with the master-slave method are simpler because it is easier to configure their control parameters and their implementation. On the other hand, the GA with the multiple communication populations allow more parallelism, but are more difficult to implement because in this type of parallelism, it is necessary to define the values of several additional parameters, besides the usual parameters of the GA itself which affect the efficiency and quality of the solutions found by this method.

In this parallelization model of genetic algorithms, a master station is dedicated to the steps: selection, crossover and mutation. The rest of the slave stations concerned with the assessment of the individuals fitness (see Figure 2). This is motivated by the fact that the 90% of the overall computation time of this method is allocated to the generated solutions fitness assessment.

Figure 2 Model of computation parallelisation



The advantages of this technique are numerous:

- The GA master-slave algorithm explores the search space in exactly the same way as a sequential GA. Therefore, the basic structures used in the design of the sequential GA remain the same.
- Its implementation is very simple.
- In many applications, the GA in master-slave mode offers significant improvements to the final results.

A master computer ensures the generation of populations of individuals using selection, crossover and mutation. Then the generated solutions are sent to the slave stations to calculate the fitness. In this way, the overall computing time is shared between all computers in the network.

This parallelisation method keeps all the characteristics of the sequential GA, without any changes in parameters already set. The only parameter that changes in the CPU parallel algorithm is the size of the population which directly depends on the number of computers on our star-shaped network, since the master computer divides the population during processing into a number, i.e. proportional to the number of computers that are available on the network, then it sends the sub-populations to the various slave computers where the calculation of the fitness is carried out. In the next section we present another parallelization paradigm based on the exploitation of the graphics processor (GPU).

4.2. Using GPU

The growth of the video game industry and multimedia applications has prompted manufacturers to produce more efficient graphics processors in terms of power computing. This development follows a different way other than the development of CPU processors. This discrepancy can be explained by the difference that exists between the categories of problems to deal with. Indeed a CPU aims to complete a task as quickly as possible against a GPU, which performs a similar treatment on multiple data during the same time and as fast as possible.

The use of this computing power outside the multimedia field is called GPGPU (General Purpose Graphics Computing Unit). To take advantage of this computing power, manufacturers have introduced tools to exploit these resources. Thus NVIDIA proposed an environment of a parallel computing architecture, which uses the parallel computing capability of NVIDIA GPUs, called CUDA (Compute Unified Device Architecture). The latter involves a compiler and a set of development tools, which can be used with the C language to write code designed to run on the GPU. To understand the idea of such an environment we must first introduce the hardware and software models. Unlike CPUs, graphics processors are optimized to perform the same treatment on a large amount of data. This difference exists in both of the memory system and the execution of the instructions.

GPUs are built in a scalar manner i.e. they have several computation units, which can carry a limited number of instructions at the same time. A GPU consists of several independent multiprocessors; each of which contains 8 Scalar Processors SP, two specialized processors and a shared memory. To manage multiple threads running several programs, the multiprocessors use a new architecture called SIMT (Single Instruction Multiple Thread) unlike SIMD (Single Instruction Multiple Data) used in early versions of CUDA. These multiprocessors connect each thread to a scalar processor SP. Then, each thread can be run independently with its own instruction address and its own registers. SIMT Multiprocessors create, manage and launch instructions on groups of 32 elements, each element being called a thread (not to be confused with a CPU thread) and each group of 32 elements is called a warp. Each multiprocessor is composed of:

- A set of 32-bit registers.

- A shared memory cache.
- A constant memory cache that is shared by all SPs (accessible in read-only).
- A cache read-only access memory called texture that is also shared by all SPs; each access to this memory area is via specific textures units.

Next, we try to identify the parts of the parallel GA algorithm that can be delegated to the GPU. First, we have used the same GA parallelization idea (Master/Slave) described in the beginning of this section, i.e. which delegates the decoding of solutions and the computing of chromosomes fitness to the GPU.

To do this, we must define the main memory space needed:

- A Storage space for the operations processing time vector. The size of the latter is n which represents the number of operations. Hence, the i^{th} vector cell represents the processing time of the i^{th} operation.
- A Storage space for the machines vector. The size of the latter is n. Hence, the i^{th} vector cell contains the machine number which will execute the i^{th} operation.
- A Storage space for the chromosomes vector (matrix chromosomes), containing the chromosomes to be treated on the GPU for decoding and evaluation.

Another storage space is needed for decoding and evaluating solutions:

- An $m \times nbc$ storage space containing, for each chromosome, the list of the ready operations that are to be performed, where, m is the number of machines and *nbc* the number of chromosomes.
- An $m \times nbc$ storage space containing, for each chromosome, the status (available or not) of each corresponding machine.
- An $m \times nbc$ storage space containing, for each chromosome, the list of the blocking operations on each machine.
- An $m \times nbc$ storage space containing, for each chromosome, the available operations to perform.
- An $m \times nbc$ storage space needed for each chromosome computing evaluation.
- An $n \times nbc$ storage space containing, for each chromosome, the scheduled operations sequence.
- An *nbc* storage space containing, chromosomes evaluation.

As we can see, the algorithm requires much memory resources. In addition, memory accesses during the solution construction phase are very important; this can lead to a long execution time. To solve this problem and reduce memory access while optimizing bandwidth, we have followed the various optimization strategies described in the CUDA programming guide. So we have made the following choices:

[1] The use of texture units to optimize bandwidth for accessing to the chromosomes matrix, operation processing time vector and the machines vector.
[7] Stocking of the remaining data in global memory.
[8] Each thread execution is a construction of a solution. In addition, this thread can access only the corresponding storage space.
[9] The thread data are aligned side by side. This alignment gives more opportunities for different threads to coalesced memory access so as to reduce the latency of global memory accesses.
[10] Access to the chromosomes data (operation processing time, appropriate machine, etc.) is done through the texture units. This allows us to harness the caching technique offered by the graphics controller, and also to increase the virtual bandwidth.

5. Results and Discussion

In this section, we present the results obtained by Tabu search using our neighbourhood function and parallel GA with CPU's and GPU's that we have developed throughout this paper.

All the presented results, for Tabu Search method have been tested on a dual-core Processor 2.10 GHz with 2 GB of Memory. The programming language used is JAVA under windows 7. However, For the Parallel GA with CUDA, the tests were performed on a workstation characterized by the following: 2×CPU's XEON E5620 2.40 Ghz; RAM 6GB (DDR2); OS Windows Seven 64 bit and GPU NVIDIA Quadro 2000 card with 01 GB.

To make the process of comparison significant, we use the so-called Lawrence Benchmarks (Lawrence, 1984) which are widely utilized in the literature. Our results are compared with the best ones, already known in the literature (Gröflin and Klinkert, 2009).

We have tested the performance of our methods on 20 instances with various sizes and difficulties. For each instance, five experiments are carried out and we select the best solution. The results obtained are summarized in Table 1.

In Table 1, the first column reports the name of the instance; SZ is the instance size, Best_S is the best known solution (Gröflin and Klinkert, 2009), TS is the makespan given by our Tabu Search method and Nb_Iter is the iteration number used in TS.

The results marked with (*) represent the instances that improve the best known solution in the literature. We notice that our Tabu search method has improved a great number of solutions. Of the forty Benchmarks tested, only four could not be improved.

It may be interesting to observe the empirical dominance relationships between the resolution methods in order to deduce a classification. We say that a metaheuristic dominates empirically another if for all instances tested, the solutions values given by the first are better or the same that those obtained by the second, with at least one strictly better solution. According to the results of Table 2, no dominance relation can be observed.

The first result to be noted is the efficiency of the metaheuristics in solving the Job Shop problem with blocking. The other result is: Despite the non-existence of no dominance relationship between our Tabu Search and Gröflin's Tabu Search with the tested examples, we can say that our Tabu Search is more appropriate according to the importance of improvements number.

When solving the job shop problem with blocking, we noticed sometimes, the existence of several different longest paths in the graph that represent the solution i.e. they all have the same the Cmax value. Thus the establishment of an effective rule to make a choice between these longest paths is necessary.

In this section, we present the influence of the selected longest path on the quality of the final solution found. Table 2 shows some instances with two different longest paths and the corresponding values of makespan.

Table 1 Tabu Search Results

Instance	SZ	Best _S	TS	Nb_Ite
la 01	10x5	832	838	5000
la 02	10x5	793	858	2000
la 03	10x5	747	725*	5000
la 04	10x5	769	756*	5000
la 05	10x5	698	671*	5000
la 06	15x5	1180	1167*	4000
la 07	15x5	1091	1076*	3000
la 08	15x5	1125	1110*	1800
la 09	15x5	1223	1172*	500
la 10	15x5	1203	1171*	400
la 11	20x5	1584	1548*	1400
la 12	20x5	1391	1378*	3700
la 13	20x5	1548	1527*	580
la 14	20x5	1620	1580*	600
la 15	20x5	1650	1632*	400
la 16	10x1	1142	1118*	6800
la 17	10x1	1026	964*	500
la 18	10x1	1078	1085	1800
la 19	10x1	1093	1085*	3200
la 20	10x1	1154	1095*	6800
la 21	15x1	1545	1600	40
la 22	15x1	1458	1455*	20
la 23	15x1	1611	1549*	40
la 24	15x1	1571	1482*	20
la 25	15x1	1499	1441*	60
la 26	20x1	2162	2069*	140
la 27	20x1	2175	2170*	200
la 28	20x1	2071	2055*	500
la 29	20x1	2124	2025*	300
la 30	20x1	2171	2088*	100
la 31	30x1	3167	3031*	600
la 32	30x1	3418	3380*	100
la 33	30x1	3131	3112*	120
la 34	30x1	3205	2962*	120
la 35	30x1	3311	3066*	150
la 36	15x1	1932	1925*	280
la 37	15x1	2058	2027*	200
la 38	15x1	1875	1732*	1500
la 39	15x1	1950	1877*	220
la 40	15x1	1936	1828*	200

Path1 is the longest path in which the number of manipulated alternative arcs is maximum, while Path2 is the longest path in which the number of handled alternative arcs is minimum. We have observed that the results obtained by choosing Path1 are always better than choosing Path2. This can be explained by the fact that: increasing the number of alternative arcs handled generates increasing number of neighboring solutions and therefore we have more chance of getting on a path that leads to a neighborhood that contains a better solution. This has allowed us to fix the iterations number of our Tabu search to the maximum number of alternative arcs changed.

Table 2 Results obtained by varying the longest path

Exa	La_0	La_0	La_0	La_0	La_1	La_1
Path1	918	880	718	1210	1245	1658
Path2	950	900	768	1280	1280	1703

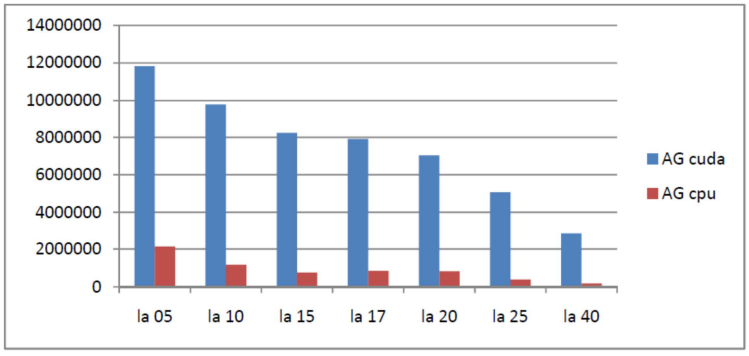
We present, in this part the gain obtained by harnessing the computing power of NVIDIA graphic cards (see figure 3).

The results of our CUDA algorithm are shown in Table 3. The first column contains the instance name; the second one contains the number of solutions explored next to the Cmax value obtained by the Genetic Algorithm in CUDA. These results have been obtained by using the configuration of 66 blocks with 16 threads per block (over a period of 300s), giving a total of 1056 threads. The third column represents the number of solutions explored next to the Cmax value obtained by the sequential Genetic Algorithm with a population size equal to 1056 and total execution time limited to 300s. One can easily notice the performance of CUDA computing in the number of explored solutions.

Table 3 The number of solutions explored by GPU

Instances	GA using CUDA		GA Using CPU	
	Number of explored solutions	*Cmax*	Number of explored solutions	*Cmax*
La_05	11827200	735	2129920	740
La_10	9793536	1320	1167360	1272
La_15	8220672	1737	757760	1763
La_17	7913664	1087	839680	1093
La_20	7028736	1239	819200	1205
La_25	5067562	1733	389120	1730
La_40	2881536	2180	184320	2184

Figure 3 Graphical representation of Table 3



6. Conclusion

Our paper provides a comparison between three methods. The first approach is an improvement of the Tabu search method presented by Gröflin and Klinbert (2009) to solve the scheduling problem with blocking. This is done by designing a new neighborhood for local search. The solutions encoding is based on alternative graphs, and the evaluation function of every schedule is based on the longest path search, that we have also improved by recalculating, only the changes generated by the moves.

This has allowed us to reduce significantly the computation time that we exploit in the exploration of a greater neighborhood. The obtained results are very satisfying because we have improved 36 instances from forty Benchmarks known in the literature. The second and the third approaches' are two parallelization techniques applied on genetic algorithm, the first of which is a master / slave method which uses a network computer and the second exploits the GPU graphics processors under CUDA. The results obtained are also very interesting.

An interesting improvement of our work in the future is to see the effect of parallelization of Tabu method proposed in this paper on the considered problem.

7. References

[1] AitZai, A., Benmedjdoub, B. and Boudhar, M. (2012) "A branch and bound and parallel genetic algorithm for the job shop scheduling problem with blocking", International Journal of Operational Research, Vol. 14, pp. 343–365.
[2] AitZai, A. and Boudhar M. (2013) 'Parallel branch-and-bound and parallel PSO for the job shop scheduling with blocking', International Journal of Operational Research, Vol. 16, pp. 14–37.
[3] AitZai, A., Dabah, A., and Boudhar, M., (2013) "Parallel CPU and GPU computations to solve the job shop scheduling problem with blocking", IEEE High Performance Extreme Computing Conference (HPEC'13), 2013
[4] Ben Mabrouk, B., Hasni, H. and Mahjoub, Z. (2009) 'On a parallel genetic–tabu search based algorithm for solving the graph colouring problem', European Journal of Operational Research, Vol. 197, pp. 1192–1201.
[5] Brizuela, C.A., Zhao, Y. and Sannomiya, N. (2001) 'No-wait and blocking job-shops: Challenging problems for GA's', IEEE 0-7803-77-2/ 01 2349-2354.
[6] Brucker, P., Jurisch, B. and Sievers, B. (1994) 'A branch and bound algorithm for the job-shop scheduling problem', Discrete Applied Mathematics, Vol. 49, pp. 107–127.
[7] Carlier, J. and Pinson E. (1989) 'An algorithm for solving the job-shop problem', Management Science, Vol. 35, pp. 164–176.
[8] Carlier, J. and Pinson, E. (1994) 'Adjustment of heads and tails for the job-shop problem', European Journal of Operational Research, Vol. 78, pp. 238–251.
[9] Dabah, A., Bendjoudi, A. and AitZai, A. (2016) Efficient parallel B&B method for the blocking job shop scheduling problem, International Conference on High Performance Computing & Simulation (HPCS).
[10] Dabah, A., Bendjoudi, A., and AitZai, A., (2017) "An efficient Tabu Search neighborhood based on reconstruction strategy to solve the blocking job shop scheduling problem", 13(4): 2015-2031 doi: 10.3934/jimo.2017029
[11] Fisher, H. and Thompson, G.L. (1963) Probabilistic learning combinations of local job-shop scheduling rules, in J.F. Muth and G.L. Thompson (Eds.), Industrial Scheduling. Englewood Cliffs: Prentice-Hall, pp. 225–251.
[12] Gorges-Schleuter, M. (1989) ASPARAGOS: an asynchronous parallel genetic optimization strategy, in Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann Publishers, pp. 422–427.
[13] Gröflin, H. and Klinkert, A. (2004) Local search in job shop scheduling with synchronization and blocking constraints, Internal working paper 04-06. Dept. of Informatics, University of Fribourg, Switzerland.
[14] Gröflin, H. and Klinkert, A. (2009) 'A new neighbourhood and tabu search for the blocking job shop', Discrete Applied Mathematics, Vol. 157, pp. 3643–3655.
[15] Gruau, F. (1994) Neural network synthesis using cellular encoding and the genetic algorithm, PhD Thesis, Ecole Normale Supérieure de Lyon.
[16] Hall, N.G. and Sriskandarajah, C. (1996) 'A survey of machine scheduling problems with blocking and no-wait process', Operations Research Vol. 44, pp. 510–525.

[17] Klinkert, A. (2001) Optimization in design and control of automated high density warehouses, Doctoral Thesis No. 1353, University of Fribourg, Switzerland.
[18] Louaqad, S. and Kamach. O. (2016) 'Mixed Integer Linear Programs for Blocking and No Wait Job Shop Scheduling Problems in Robotic cells', International Journal of Computer Applications (0975 - 8887), Vol. 153 - No. 10.
[19] Lawrence, S. (1984) Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Carnegie Mellon Univ, Pittsburgh, Pennsylvania.
[20] Manderick, B. and Spiessens, P. (1989) Fine-grained parallel genetic algorithms, in J. David Schaffer (Ed.). In Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann Publishers, pp. 428–433.
[21] Mascis, A. and Pacciarelli, D. (2002), Job shop scheduling with blocking and no-wait constraints, European Journal of Operational Research, Vol. 143, pp. 498–517.
[22] Mati, Y. Rezg, N. and Xie, X. (2001) 'A taboo search approach for deadlock-free scheduling of automated manufacturing systems', Journal of Intelligent Manufacturing, Vol. 12, pp. 535–552.
[23] Mühlenbein, H. (1989) Parallel genetic algorithms, population genetics and combinatorial optimization. Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann Publishers, pp. 416–421.
[24] Perregaard, M. and Clausen, J. (1998) 'Parallel branch-and-bound methods for the job-shop scheduling problem', Annals of Operations Research, Vol. 83, pp. 137–160.
[25] Pham, D. and Klinkert, A. (2008) 'Surgical case scheduling as a generalized job shop scheduling problem', European Journal of Operational Research, Vol. 185, pp. 1011–1025.
[26] Pranzo, M. and Pacciarelli. D. (2016) 'An iterated greedy metaheuristic for the blocking job shop scheduling problem', Journal of Heuristics, Vol. 22, No. 11, pp. 587–611.