© International Journal of Combinatorial Optimization Problems and Informatics, Vol. 10, No. 1, Jan-April 2019, pp. 6-15. ISSN: 2007-1558.

A Hybrid Simulated Annealing for Job Shop Scheduling Problem

Leonor Hernández-Ramírez[1], Juan Frausto-Solis[1], Guadalupe Castilla-Valdez[1], Juan Javier González-Barbosa[1], David Terán-Villanueva[1], M. Lucila Morales-Rodríguez[1]

[1]*TECNM/INSTITUTO TECNOLÓGICO DE CIUDAD MADERO*

iscleo1@gmail.com, juan.frausto@gmail.com, gpe_cas@yahoo.com.mx, jjgonzalezbarbosa@hotmail.com, david_teran01@yahoo.com.mx, lmoralesrdz@gmail.com

The Job Shop Scheduling Problem (JSSP) arises in the context of high-performance computing and belongs to the NP-hard combinatorial optimization problems. The purpose of JSSP is to find the order of execution of a set of jobs on a group of machines, subject to certain precedence and resource availability constraints. The objective in this problem is minimizing the *makespan* that is the time elapsed from the starting time of the first job until the completion time of the last job. In this paper, a novel hybrid algorithm named AntGenSA for solving JSSP is proposed. AntGenSA uses Ant Colony System (ACS), Simulated Annealing (SA), and Genetic Algorithm (GA). To assess the performance of this algorithm, it is executed in a parallel computer, using a set of instances proposed by Fisher-Thompson, Yamada-Nakano, Taillard, Lawrence, and Applegate-Cook. The evaluation of this algorithm was performed mainly by the quality of the solution but the execution time was measuring as well. The experimental results show that the performance of the parallel execution of AntGenSA is highly competitive with the state-of-the-art algorithms.

Keywords: JSSP, Simulated Annealing, Ant Colony System, Genetic Algorithm, OpenMP.

1. Introduction

The objective of the JSSP is to find an optimal sequence for executing a finite set of operations without violating constraints. JSSP is common in the manufacturing industry; an optimal solution reduces lead times, costs, and optimizes machine utilization. JSSP is one of the most difficult NP-hard problems. Consequently, exact methods (as Branch and Bound) cannot solve it in an efficient way; because these algorithms may require a prohibitive processing time for solving JSSP instances. For this reason, heuristic methods are used, which provide approximate solutions to the optimum in reasonable times.

JSSP has been solved using a wide range of metaheuristic algorithms; nevertheless, all of them have strengths and weaknesses [1]. The current tendency for solve JSSP is to combine two or more metaheuristics; in this way, weaknesses of one method can be supplemented by the strengths of another interacting synergistically to search the optimal solution. Experiments show that these hybrid methods perform better than their corresponding single methods because they help each other to escape from local optima [1]. Tabu Search with Path Relinking (TS/PR) is an example of hybrid algorithms for JSSP [2]. This algorithm solved a JSSP instance that had remained unsolved for twenty years [2]. Furthermore, another hybrid algorithm based on Simulated Annealing (SA) surpassed TS/PR by using a semi-local search method and a Cauchy's probability density function [3]. This algorithm was evaluated using 88 instances obtaining an excellent performance with instances taken from Fisher-Thompson [4]; Lawrence [5], and Applegate-Cook[6]. These instances are frequently used for comparative analysis of JSSP algorithms. In addition, let us to mention that other instances taken for testing JSSP algorithms are those published by Yamada & Nakano [7] and Barnes and Chambers [8]. Even though the efficiency of SA for JSSP, only a few publications use the parallel approach [9]. In contrast, there are some hybrid parallel algorithms based on Tabu Search (TS), tested with Barnes-Chambers instances and implemented with CUDA and MPI [9]. Another hybrid parallel algorithm named Neuro-Tabu based on TS and neural networks for JSSP was published in 2013 [10]; in this algorithm, MPI was used to distribute the calculations of the GPUs. A very natural

Received May 24, 2016 / Accepted Sep 11, 2017

way to implement hybrid parallel algorithms is the use of Ant Colony Optimization (ACO) with some metaheuristics; this kind of hybridizations algorithms can improve the performance of the individual metaheuristics[11]. Besides, it is well known that Genetic Algorithms obtain very good results when they are used in parallel implementations for JSSP.

There are not only theoretical reasons for using a hybrid approach when JSSP algorithms are designed; in addition, experimental results show that the hybrid algorithms perform better than its corresponding individual model. This is because the hybridization convergence rate is usually high and it helps to escape from local optima[1]. Therefore, researchers are constantly searching new algorithms and a lot of efforts have been made to improve existing methods.

In this work a hybrid metaheuristic algorithm Ant Colony System/Genetic Algorithm/Simulated Annealing (AntGenSA) for solving JSSP is proposed. This algorithm uses Simulated Annealing because this metaheuristic has obtained competitive results for JSSP [3], uses population algorithms (Ant Colony and Genetic Algorithm) which have the characteristic of working with several solutions at the same time and this was used to develop the parallel model for the proposed algorithm.

The paper is organized as follows: In section 2, a general background is presented and the JSSP formulation and the disjunctive graph representation are explained. In section 3, the sequential and the parallel variants of the proposed algorithm are presented. These algorithms are a hybridization of Ant Colony System, GA, and SA. In section 4 the proposed algorithms are evaluated using a set of benchmark instances. In this section, sequential and parallel versions are compared considering the state-of-art algorithms as a reference. Finally, general conclusions are presented at the end of the paper.

2. Background

In this section is shown the theoretical foundations on which this paper is sustained.

2.1 The Job Shop Scheduling Problem (JSSP)

We have a set of *J* jobs to be processed in a set of *M* machines, in a previously established order, under given constraints. The objective of JSSP is to organize these jobs optimally without violating any restrictions. In this problem, there are two types of constraints: a) *Sequence constraints*, which indicate that the precedence relationships between the operations of a job must be guaranteed; b) *Constraints of resources* indicate that no more than one job can be executed on a machine at the same time.

A scheduling problem is considered completely solved if the starting times of all operations are determined and the sequence and resource constraints are not violated [10] [11] [12]. The size of the JSSP is defined by $j \times m$, where *j* is the jobs number in $J = \{J_1, J_2, J_3, \dots J_j\}$ and *m* is the machines number of the problem in $M = \{M_1, M_2, M_3, \dots M_m\}$. Each element of the set *J* must be related with an element of the set *M* of machines. The relation that associates the jobs in *J* with the machines in *M* conforms a binary relation J→*M*, called *Operation* (*j*,*m*) [13]. The total number of operations of the problem is denoted by $O_{JM} = \{O_{1,1}, O_{1,2}, O_{1,3}, \dots O_{j,m}\}$, where $O_{JM} = \{(j,m) | j \in J, m \in M\}$. The set of operations *O* corresponding to the same job has a processing sequence called technological sequence *S*, such that $S = \{S_1, S_2, S_3, \dots S_m\}$ where *m* is the number of machines in the problem. Therefore, each job *j* has the technological sequence: $O_{JS} = \{O_{j,1}, O_{j,2}, O_{j,3}, \dots O_{j,s}\}$. Each operation (*j*, *m*) has associated a processing time (*p*) that indicates the required time of that *j* job in the *m* machine.

In summary, JSSP is defined with:

Set of machines	$M = \{M_1, M_2, M_3, \dots M_m\}$
Set of jobs	$J = \{J_1, J_2, J_3, \dots J_j\}$
Set of operations	$O_{JM} = \{O_{1,1}, O_{1,2}, O_{1,3}, \dots O_{j,m}\}$
Technological sequence	$O_{JS} = \{O_{j,1}, O_{j,2}, O_{j,3}, \dots O_{j,s}\}$

Each operation $i \in O$ is linked to a job $j_i \in J$ to which belongs and a machine $m_i \in M$ in which it must be done, consuming an uninterrupted time $p_i \in \mathbb{R}$. In addition, we have a binary precedence relation PREC that partitioning *O* in sequences, one for each job. JSSP is to find a start time t_i for each operation $i \in O$ trying to minimize the *makespan*. There are several variants of JSSP depending on the objective; the most common is to minimize the *makespan*, which is the completion time of the last task subject to the established precedence constraints. The *makespan* is denoted by C_{max} and is the objective function in this paper.

Job Shop Scheduling Problem consists in:

Minimize $C_{max} = Max\,(t_i + p_i): \forall J_i \in J,\ M_i \in M$

Subject to:

$t_j \geq t_i + p_i$ For all $i, j \in O$ with *i* PREC *j* (1)

$t_j \geq t_i + p_i\ o\ t_i \geq t_j + p_j$ For all $i, j \in O$ with $m_i = m_j$ (2)

Where:

t: Start time of task *i* and p_i is its duration.

C_{max}: *Makespan* of the set of tasks.

PREC: Indicates that the operations of a job must be done in a precise order for the problem, such that if *i* PREC *j* then *j* cannot start before *i* ends.

Constraint (1) indicates the precedence relation of the operations, that the operations of a job must be performed in a precise order for the problem, such that if *i* PREC *j* then j cannot start before *i* ends. Constraint (2) implies that the machine can process only one operation at the same time.

2.2 Complexity of the problem

There are many combinatorial optimization problems whose models are relatively easy to build, but the algorithms to find the best solutions are not easy to define. This is the case of JSSP, that as we mentioned before this problem belongs to NP-hard Problem class [14]. In fact, it is too difficult to find a very good solution for many instances of JSSP. In addition, so far, there is not a deterministic algorithm that solves the problem in a polynomial time. Consequently, to design heuristic methods is up to now the best alternative to solve them. A good representation of the problem useful for design this algorithm is the disjunctive graph presented in the next section.

2.3 Graphical representation of the problem

The Job Shop Scheduling Problem (JSSP) can be formally represented by a disjunctive graph G = (V; C ∪ D) [7]. Where:

- V is a set of nodes that represents operations of the jobs with special nodes, the source and the sink nodes. In Figure 1, these are identified with a "0" and a "*" respectively.
- C is a set of conjunctive arcs that represents the technological sequences of machines for each job.
- D is a set of disjunctive arcs that represents pairs of operations that must be performed on the same machine.

The disjunctive graph showed at Figure 1 and taken from Yamada has 3 tasks [7]. In this graph, each task has several operations that are numbered within the circles of this figure. Besides, source and sink nodes represent dummy operations but should be added to the graph to identify the starting and final states of JSSP. In Figure 1, the processing time for each operation is the value written aside all nodes except the dummy operations because their processing tine is zero. The arrows of the disjunctive graph represent precedence's constraints that must be satisfied. There are two types of arrows [7]: a) Arrows in a single direction and for which there is no doubt that operation *i* precedes operation *j*; b) Arrows in two directions and for which the precedence can be presented in a way that *i* precedes j or j precedes *i*.

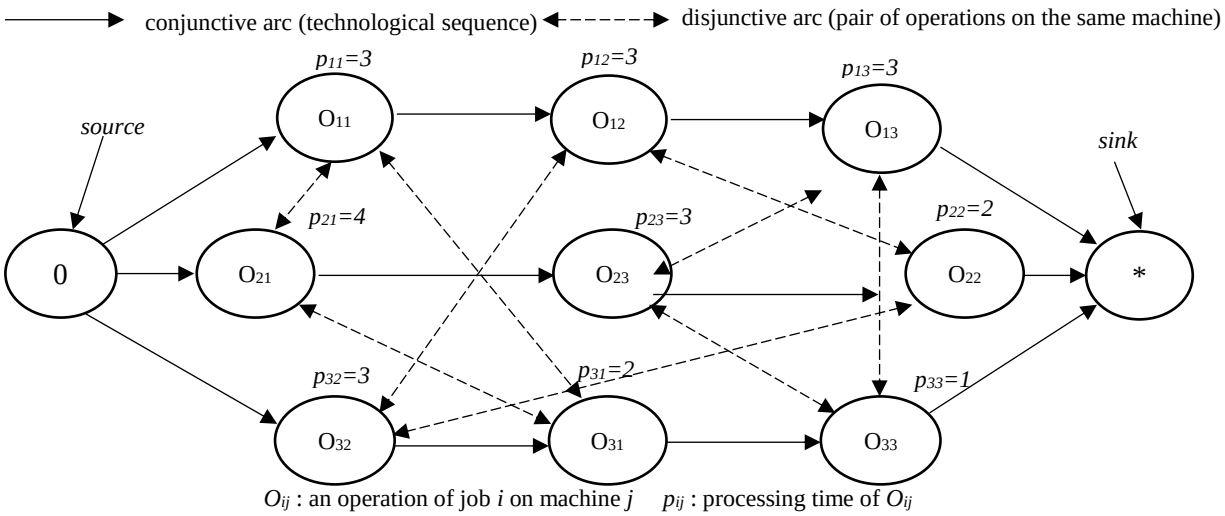

Figure 1. Disjunctive graph of a 3x3 problem.

3 AntGenSA: The Proposed Algorithm

SA was developed by Cerny and Kirkpatrick adapting strategies from the field of the statistical mechanics for solving the combinatorial optimization problems [15] , [16]. In these problems the minimum value of a function that depends on several parameters is sought. SA explores the neighborhood of the current randomly generated solution searching for the best solution; the new solution is accepted if its fitness is lower than the current solution fitness; however, worse solutions still could be accepted with a probability of acceptance *p* determined by the Boltzmann distribution. The Boltzmann probability of acceptance decreases as the temperature decreases.

To solve the problem a hybrid algorithm based on Simulated Annealing (SA) with Ant Colony and Genetic Algorithm was developed (AntGenSA). The AntGenSA method generates a set of *n* solutions (initial population) using Ant System algorithm. These solutions evolve along the algorithm iterations. Afterward, an iterative process randomly selects two solutions from the *n* initial solutions at each iteration that will be the parent solutions, and then go in the SA outermost cycle controlled by the final temperature parameter. Inside this cycle controlled by temperature, a Markov perturbation cycle is performed using the crossover genetic operator. The parent solutions are combined to produce two offspring solutions with a crossover operation. In the acceptance phase, each child is compared with a parent through its fitness value, if it is better than its parent, then the corresponding child replaces to its parent. Otherwise, the child has another replacing chance according to the Boltzmann probability. When the algorithm reaches the final temperature, the current temperature is reset and two new solutions are selected. The process continues until all the solutions are processed. The parallelization model designed for AntGenSA uses independent cycles for each pair of solutions that evolve through Markov perturbations and the genetic operators allows an efficient distribution in the processing threads used. In this way, it is possible to reduce the processing time of the algorithm, which now solely depends on the time that it takes to process the thread that requires the longest processing time. For parallelization were used OpenMP directives. In Figure 2 AntGenSA is shown:

```
1.   sOlds[] ← Generate initial population with Ant System
2.   makespanOld(Ki) ← CalculateMakespan(Ant(Ki))
3.   BestMakespan = Min(MakespanOld(Ki)) // Get best Makespan
4.   For i = 0 to K do (i++2)
5.     While(Tcurrent>Tfinal) do
6.       For j = 0 to Lk  do
7.          sNews[i] ← Generate 2 news ants with crossover selecting sOlds(2/2)
8.          MakespanNew(Ki) ← CalculateMakespan(ant(Ki))
9.          For m = i to i+2 do
10.            Δ Em= MakespanNew(Km) - MakespanOld(Km)
11.            if (Δ Em<0) then
12.               sOlds(Km) ←sNews(Km)
13.               MakespanOld(Km) ← MakespanNew(Km)
14.               BestMakespan = Min{BestMakespan, Min((MakespanNew(Km)}
15.            else
16.               r← random(0,1)
17.               if (r<e -ΔEm/Tcurrent) then
18.                  sOld(Km) ← sNew(Km)
19.                  MakespanOld(Km) ← MakespanNew(Km)
20.               endif
21.            endif
22.         endFor
23.      endFor
24.      Tcurrent = α*Tcurrent
25.    EndWhile
26.    Tcurrent = Tinitial
27.  EndFor
28.  Return sOlds[],MakespanOld[], sOlds*, MakespanOld*, BestMakespan
```

Figure 2. AntGenSA Algorithm.

4 Experimental Design

The methodology applied in experimentation, as well as the characteristics of hardware and software used in the experiments are described in this section.

To assess the algorithm performance 25 instances were used. These were taken from five sources:

- ft06, ft10 and ft20 [4]
- yn1, yn2, yn3 y yn4 [7]
- ta01, ta11, ta21, ta31, ta41, ta51, ta61 and ta71 [17]
- la01, la06, la16, la21 and la26 [5]
- orb01, orb02, orb03, orb04 and orb05 [6]

Each instance was solved using different processing time according to its size, that is, for smaller instances, the used processing time was shorter than the processing time used for larger instances. For each instance, the same processing time was used with each one of the assessed algorithms. That is, the sequential and parallel versions.

The following performance indicators were assessed as follows:

a) Runtime expressed in minutes
b) Quality of the solution; measured by the *makespan* value.

To compare the performance in time and quality, the average was calculated, also the best solution (lowest *makespan)* obtained by each algorithm was recorded. Results are shown in next section.

In order to improve the performance of the AntGenSA, a new variant of the hybrid algorithm was implemented. Given that during the tests for tuning the algorithm's parameters, it was observed that the crossover process was highly resource consuming. Then it was considered the possibility of omitting this process so that the algorithm would expand the search in the regions close to good solutions.

Basically, this new variant of the algorithm omits the crossover process in order to improve performance, reducing exploration diversification with and increasing intensification. It should be mentioned that the algorithm maintains a good level of diversification because it still has processes aimed at diversifying, such as Boltzmann's probabilistic function of simulated annealing, as well as genetic mutation.

Both programs, the sequential and parallel program were implemented in C language. Parallel version was developed using OpenMP directives. The execution of the programs was made in a terminal of the Ehecatl cluster of the Technological Institute of Ciudad Madero, with the following characteristics: Intel® Xeon® processor at 2.30 GHz, Memory 64GB (4x16GB) ddr4-2133 and Linux CentOS operating system.

4.1 Results

Table 1 shows the solutions average and the best solution obtained for AntGenSA compared with the best known solution (BKS) for each instance. Columns 1, 2 and 3 show the instance name, instance size and his BKS, next columns show results obtained by AntGenSA with crossover in sequential and parallel versions and without crossover respectively.

In Table 1, the smallest error values for each instance have been marked. Notice that the parallel version of AntGenSA without crossover obtains the best results; it obtains the smaller percentage of error in twenty-one instances. On the other hand, the sequential version of AntGenSA using crossover obtains the worst results; this version only achieves the smaller errors in four instances.

Table 2 has the same content as table 1 in columns 1, 2 and 3. In the next eight columns, the results obtained for two algorithms of the state of the art are shown. It shows results obtained by a hybrid Tabu Search/Path Relinking (TS/PR) [2], and for a Hybrid Algorithm of Fast Simulated Annealing with Quenching (HFSAQ) [3]. Last columns show results obtained for the version of the AntGenSA without crossover.

In Table 2 have been marked eight values that overcome or ties the results obtained by the algorithms of the state-of-the-art. Notice that AntGenSA obtains fourteen solutions with error less than 1% respect to the best known solution (BKS), with error less than 2 % respect to BKS and twenty-three with error less than 3 % respect to BKS. In general AntGenSA algorithm obtains competitive results in comparison with the state of the art algorithms. Notice that obtains an average relative error (ARE) of 1.1% for the twenty five analyzed instances.

Table 1. Results with AntGenSA with crossover and without crossover.

			AntGenSA with Crossover								**AntGenSA without Crossover**							
			Sequential				**Parallel**				**Sequential**				**Parallel**			
			Average		**Best**		**Average**		**Best**		**Average**		**Best**		**Average**		**Best**	
Instance	**Size**	**BKS**	**MKS**	**RE (%)**	**MKS**	**RE (%)**	**MKS**	**RE (%)**	**MKS**	**RE (%)**	**MKS**	**RE (%)**	**MKS**	**RE (%)**	**MKS**	**RE (%)**	**MKS**	**RE (%)**
ft06	6x6	**55**	55.0	0.0	55	0.0	55.0	0.0	55	0.0	55.0	0.0	55	0.0	55.0	0.0	55	0.0
ft10	10x10	**930**	955.3	2.7	950	2.2	954.5	2.6	938	0.9	952.0	2.4	947	1.8	945.0	1.6	938	0.9
ft20	20x5	**1165**	1177.3	1.1	1176	0.9	1169.0	0.3	1165	0.0	1178.9	1.2	1167	0.2	1170.5	0.5	1165	0.0
yn01	20x20	**884**	910.8	3.0	907	2.6	906.3	2.5	899	1.7	910.3	3.0	901	1.9	898.7	1.7	896	1.4
yn02	20x20	**904**	949.3	5.0	944	4.4	937.3	3.7	922	2.0	956.7	5.8	952	5.3	924.3	2.2	915	1.2
yn03	20x20	**892**	918.0	2.9	918	2.9	909.0	1.9	904	1.3	908.5	1.8	904	1.3	905.0	1.5	900	0.9
yn04	20x20	**968**	1035.0	6.9	1031	6.5	1020.5	5.4	1012	4.5	991.7	2.4	988	2.1	988.3	2.1	984	1.7
ta01	15x15	**1231**	1301.1	5.7	1290	4.8	1264.5	2.7	1256	2.0	1287.3	4.6	1269	3.1	1251.3	1.6	1241	0.8
ta11	20x15	**1357**	1465.0	8.0	1453	7.1	1428.3	5.3	1419	4.6	1405.9	3.6	1398	3.0	1398.5	3.1	1383	1.9
ta21	20x20	**1642**	1734.7	5.6	1725	5.1	1722.5	4.9	1710	4.1	1685.5	2.6	1685	2.6	1679.0	2.3	1671	1.8
ta31	30x15	**1764**	1865.6	5.8	1854	5.1	1804.5	2.3	1794	1.7	1870.3	6.0	1863	5.6	1767.3	0.2	1766	0.1
ta41	30x20	**2005**	2342.0	16.8	2342	16.8	2231.5	11.3	2227	11.1	2085.0	4.0	2076	3.5	2081.0	3.8	2072	3.3
ta51	50x15	**2760**	3169.0	14.8	3102	12.4	3163.3	14.6	3139	13.7	2920.0	5.8	2920	5.8	2762.5	0.1	2760	0.0
ta61	50x20	**2868**	3230.0	12.6	3230	12.6	2932.0	2.2	2929	2.1	2868.0	0.0	2868	0.0	2868.0	0.0	2868	0.0
ta71	100x20	**5464**	6194.0	13.4	6130	12.2	6154.2	12.6	6130	12.2	5464.0	0.0	5464	0.0	5464.0	0.0	5464	0.0
orb01	10x10	**1059**	1085.5	2.5	1059	0.0	1081.7	2.1	1059	0.0	1079.3	1.9	1059	0.0	1077.8	1.8	1059	0.0
orb02	10x10	**888**	898.7	1.2	889	0.1	895.5	0.8	889	0.1	889.7	0.2	889	0.1	890.2	0.2	889	0.1
orb03	10x10	**1005**	1044.3	3.9	1024	1.9	1039.1	3.4	1018	1.3	1023.9	1.9	1017	1.2	1022.4	1.7	1005	0.0
orb04	10x10	**1005**	1018.4	1.3	1006	0.1	1015.3	1.0	1006	0.1	1012.4	0.7	1005	0.0	1013.7	0.9	1005	0.0
orb05	10x10	**887**	894.1	0.8	890	0.3	893.0	0.7	889	0.2	893.0	0.7	887	0.0	891.4	0.5	887	0.0
la01	10x5	**666**	666.0	0.0	666	0.0	666.0	0.0	666	0.0	666.0	0.0	666	0.0	666.0	0.0	666	0.0
la06	15x5	**926**	926.0	0.0	926	0.0	926.0	0.0	926	0.0	926.0	0.0	926	0.0	926.0	0.0	926	0.0
la16	10x10	**945**	975.7	3.3	946	0.1	950.2	0.6	946	0.1	947.1	0.2	945	0.0	950.2	0.6	945	0.0
la21	15x10	**1046**	1071.9	2.5	1059	1.2	1076.7	2.9	1053	0.7	1073.2	2.6	1046	0.0	1053.4	0.7	1046	0.0
la26	20x10	**1218**	1218.0	0.0	1218	0.0	1218.0	0.0	1218	0.0	1218.0	0.0	1218	0.0	1218.0	0.0	1218	0.0

AntGenSA: Ant System/Genetic/Simulated Annealing

BKS: Best Known Solution

MKS: Makespan

RE: Relative Error

Table 2. Results with TS/PR, HFSAQ and AntGenSA parallel version without crossover.

Instance	Size	BKS	TS/PR				HFSAQ				AntGenSA Parallel without crossover			
			Average		Best		Average		Best		Average		Best	
			MKS	RE (%)	MKS	RE (%)	MKS	RE (%)	MKS	RE (%)	MKS	RE (%)	MKS	RE (%)
ft06	6x6	**55**	55.0	0.0	55	0.0	55.0	0.0	55	0.0	55.0	0.0	55	0.0
ft10	10x10	**930**	930.0	0.0	930	0.0	932.4	0.3	930	0.0	945.0	1.6	938	0.9
ft20	20x5	**1165**	1165.0	0.0	1165	0.0	1167.6	0.2	1165	0.0	1170.5	0.5	1165	0.0
yn01	20x20	**884**	885.5	0.2	884	0.0	-	-	-	-	898.7	1.7	896	1.4
yn02	20x20	**904**	907.7	0.4	904	0.0	-	-	-	-	924.3	2.2	915	1.2
yn03	20x20	**892**	893.8	0.2	892	0.0	-	-	-	-	905.0	1.5	900	0.9
yn04	20x20	**968**	969.1	0.1	968	0.0	-	-	-	-	988.3	2.1	984	1.7
ta01	15x15	**1231**	-	-	-	-	-	-	-	-	1251.3	1.6	1241	0.8
ta11	20x15	**1357**	-	-	-	-	1370.4	1.0	1361	0.3	1398.5	3.1	1383	1.9
ta21	20x20	**1642**	-	-	-	-	1652.7	0.7	1646	0.2	1679.0	2.3	1671	1.8
ta31	30x15	**1764**	-	-	-	-	1775.0	0.6	1767	0.2	1767.3	0.2	1766	0.1
ta41	30x20	**2005**	-	-	-	-	2035.7	1.5	2022	0.8	2081.0	3.8	2072	3.3
ta51	50x15	**2760**	-	-	-	-	-	-	-	-	2762.5	0.1	2760	0.0
ta61	50x20	**2868**	-	-	-	-	-	-	-	-	2868.0	0.0	2868	0.0
ta71	100x20	**5464**	-	-	-	-	-	-	-	-	5464.0	0.0	5464	0.0
orb01	10x10	**1059**	1059.0	0.0	1059	0.0	1060.0	0.1	1059	0.0	1077.8	1.8	1059	0.0
orb02	10x10	**888**	888.0	0.0	888	0.0	888.6	0.1	888	0.0	890.2	0.2	889	0.1
orb03	10x10	**1005**	1005.0	0.0	1005	0.0	1005.0	0.0	1005	0.0	1022.4	1.7	1005	0.0
orb04	10x10	**1005**	1005.0	0.0	1005	0.0	1005.4	0.0	1005	0.0	1013.7	0.9	1005	0.0
orb05	10x10	**887**	887.0	0.0	887	0.0	887.6	0.1	887	0.0	891.4	0.5	887	0.0
la01	10x5	**666**	666.0	0.0	666	0.0	666.0	0.0	666	0.0	666.0	0.0	666	0.0
la06	15x5	**926**	926.0	0.0	926	0.0	926.0	0.0	926	0.0	926.0	0.0	926	0.0
la16	10x10	**945**	945.0	0.0	945	0.0	945.0	0.0	945	0.0	950.2	0.6	945	0.0
la21	15x10	**1046**	1046.0	0.0	1046	0.0	1046.8	0.1	1046	0.0	1053.4	0.7	1046	0.0
la26	20x10	**1218**	1218.0	0.0	1218	0.0	1218.0	0.0	1218	0.0	1218.0	0.0	1218	0.0
ARE				0.1				0.3				1.1		

TS/PR: Tabu Search/Path Relinking

HFSAQ: Hybridization of Fast Simulated Annealing with Quenching

ARE: Average Relative Error

Table 3 has the same content as table 1 and 2 in the first two columns; the last columns show runtimes and percentage of improvement for the sequential and parallel algorithm with crossover and without crossover.

Table 3. Runtime for AntGenSA sequential and parallel with crossover and without crossover

Instance	Size	Runtime (Minutes) With Crossover		Improvement (%)	Runtime (Minutes) Without Crossover		Improvement (%)
		Sequential	Parallel		Sequential	Parallel	
ft06	6x6	0.11	0.09	23.93	0.13	0.11	15.53
ft10	10x10	9.28	6.79	26.81	3.02	1.72	42.83
ft20	20x5	7.72	5.87	23.96	2.11	1.97	6.67
yn01	20x20	263.10	216.59	17.68	84.48	64.79	23.31
yn02	20x20	243.25	197.25	18.91	86.23	77.58	10.03
yn03	20x20	277.70	130.09	53.15	98.04	67.57	31.07
yn04	20x20	245.86	157.23	36.05	102.01	78.90	22.65
ta01	15x15	46.47	39.27	15.50	20.05	12.54	37.44
ta11	20x15	99.80	68.27	31.60	41.87	38.39	8.32
ta21	20x20	217.19	138.52	36.22	103.05	65.00	36.92
ta31	30x15	303.43	263.74	13.08	76.61	83.75	-9.33
ta41	30x20	665.91	328.52	50.67	231.05	183.62	20.53
ta51	50x15	1027.80	653.98	36.37	247.83	141.16	43.04
ta61	50x20	2459.60	1494.90	39.22	524.71	302.46	42.36
ta71	100x20	2389.46	1263.50	47.12	1172.98	838.15	28.54
orb01	10x10	5.74	2.99	47.89	3.87	3.36	13.18
orb02	10x10	5.07	2.52	50.18	3.81	3.09	18.73
orb03	10x10	5.50	2.74	50.09	4.12	3.54	14.20
orb04	10x10	5.08	2.58	49.28	3.74	3.07	17.74
orb05	10x10	6.10	3.15	48.39	4.02	3.43	14.73
la01	10x5	0.61	0.32	47.64	0.37	0.30	17.52
la06	15x5	0.70	0.33	52.39	0.19	0.17	7.57
la16	10x10	5.07	2.54	49.81	2.87	2.67	7.05
la21	15x10	13.04	5.32	59.19	7.19	6.61	8.06
la26	20x10	28.53	10.26	64.04	11.39	10.35	9.15

The parallel version of AntGenSA without crossover obtains smaller processing time in sixteen of the twenty-five analyzed instances; on the other hand the parallel version with crossover only achieves a smaller processing time in nine instances. For example, notice that for the instance ta51 the runtime was reduced from 653.98 minutes in the crossover-sequential version up to 141.16 minutes in the parallel version without crossover. In general, the AntGenSA parallel version uses less processing time than sequential version.

5 Conclusions

This paper has presented AntGenSA, a novel hybrid simulated annealing algorithm for JSSP in sequential and parallel version; AntGenSA includes Ant Systems and Genetic Algorithms into the Simulated Annealing algorithm. The new algorithm takes advantages of the three metaheuristics and maintains the simple structure of the classical Simulated Annealing. Furthermore, the proposed algorithm uses three metaheuristic algorithms that had not been used for the JSSP solution.

The experimentation shows that the crossover operator implemented for the generation of new solutions uses a lot of processing time. By omitting this operator, the execution time of the algorithm is reduced and AntGenSA obtains better results in terms of quality; this is because the *makespan* improves considerably with respect to the version of the algorithm which uses this operator. In addition, with the parallel version without the crossover operator, the runtime is reduced as well.

Finally experimental results show that the performance of parallelized AntGenSA is highly competitive with the state-of-the-art algorithms. AntGenSA obtains an average relative error value of 1.1 % respect to the best known solution of the analyzed instances, which are commonly used to test algorithms.

Acknowledgments

The authors would like to acknowledge to CONACYT and TECNM for their support. We acknowledge to Laboratorio Nacional de Tecnologías de la Información del Instituto Tecnológico de Ciudad Madero for let us access the cluster. This work has been partially supported by CONACYT Project 404048. Finally, the authors would like to thank CONACYT for all the received support.

References

[1] X. Qiu and H. Y. K. Lau, An AIS-based hybrid algorithm for static job shop scheduling problem, *J. Intell. Manuf.*, vol. 25, no. 3, pp. 489–503, Jun. 2012.

[2] B. Peng, Z. Lü, and T. C. E. Cheng, A tabu search/path relinking algorithm to solve the job shop scheduling problem, *Comput. Oper. Res.*, vol. 53, pp. 154–164, 2015.

[3] K. Akram, K. Kamal, and A. Zeb, Fast simulated annealing hybridized with quenching for solving job shop scheduling problem, *Appl. Soft Comput. J.*, vol. 49, pp. 510–523, 2016.

[4] C. Fisher and G. L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in *Industrial Scheduling*, Englewood Cliffs: Prentice-Hall, 1963, pp. 225–251.

[5] S. Lawrence, Resource constrained project scheduling–A computational comparison of heuristic scheduling technique, Pittsburgh, Pennsylvania, 1985.

[6] D. Applegate and W. Cook, A Computational Study of the Job-Shop Scheduling Problem, *ORSA J. Comput.*, vol. 3, no. 2, pp. 149–156, May 1991.

[7] T. Yamada, Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems, Kyoto University, 2003.

[8] J. Barnes and J. Chambers, Flexible job shop scheduling by tabu search, Austin, 1996.

[9] W. Bozejko, M. Uchronski, and M. Wodecki, Parallel hybrid metaheuristics for the flexible job shop problem, *Comput. Ind. Eng.*, vol. 59, no. 2, pp. 323–333, 2010.

[10] W. Bozejko, M. Uchronski, and M. Wodecki, Parallel Neuro-Tabu Search Algorithm for the Job Shop Scheduling Problem, Springer Berlin Heidelberg, 2013, pp. 489–499.

[11] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge Ma: MIT Press, 2004.

[12] W. Bozejko, *A new class of parallel scheduling algorithms*. Poland: Oficyna Wydawnicza Politechniki Wrocławskiej, 2010.

[13] S.-C. Lin, E. D. Goodman, and W. F. Punch, Investigating parallel genetic algorithms on job shop scheduling problems, Springer Berlin Heidelberg, 1997, pp. 383–393.

[14] M. R. Garey, D. S. Johnson, and R. Sethi, The Complexity of Flowshop and Jobshop Scheduling, *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by Simulated Annealing, *Sci. New Ser.*, vol. 220, no. 4598, pp. 671–680, 1983.

[16] V. Cerny, Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm I, *J. Optim. Theory Appl.*, vol. 45, no. 41–45, 1985.

[17] É. Taillard, Benchmarks for basic scheduling problems, *Eur. J. Oper. Res.*, vol. 64, pp. 278–285, 1993.