Dynamic Seed Genetic Algorithm to Solve Job Shop Scheduling Problems

Flávio Grassi, Pedro Henrique Triguis Schimit(✉), and Fabio Henrique Pereira

Universidade Nove de Julho, São Paulo, Brazil
{schimit, fabiohp}@uninove.br

Abstract. This paper proposes a simple implementation of genetic algorithm with dynamic seed to solve deterministic job shop scheduling problems. The proposed methodology relies on a simple indirect binary representation of the chromosome and simple genetic operators (one-point crossover and bit-flip mutation), and it works by changing a seed that generates a solution from time to time, initially defined by the original sequencing of the problem addressed, and then adopting the best individual from the past runs of the GA as the seed for the next runs. The methodology was compared to three different approaches found in recent researches, and its results demonstrate that despite not finding the best results, the methodology, while being easy to be implemented, has its value and can be a starting point to more researches, combining it with other heuristics methods that rely in GA and other evolutionary algorithms as well.

Keywords: Genetic Algorithms · Job shop · Scheduling · Dynamic seed

1 Introduction

Scheduling is the process of assigning one or more resources to perform certain activities whose processing will require a certain amount of time [1]. These resources in an industrial environment may be associated with machines, and the activities that will be processed in a machine are known as operations or tasks. Thus, a job is a set of one or more tasks.

The scheduling problems in scenarios of job shop (JSSP) is a NP-hard problem which has been studied by using exact methods, such as the Branch and Bound [2] and the Shifting Bottleneck [3]. It consists of scheduling a set of tasks in different machines; known as jobs, where the precedence of each task must be obeyed. Each job can be processed in one machine at a time, and the task started in a certain machine must be completed before this resource starts a new process, i.e., no pre-emption is allowed. The time need for each task to be proceeded is known in advance for deterministic problems, and finally, all the jobs are available in the time zero [1]. Usually, researchers consider the minimization of the makespan as the goal, where makespan refers to the total time needed to process all operations of all the jobs in the machines.

In recent years, the metaheuristic methods have been widely used in solving this type of problem. Among these methods, those with greater emphasis for solving JSSP

I. Nääs et al. (Eds.): APMS 2016, IFIP AICT 488, pp. 170–177, 2016.
DOI: 10.1007/978-3-319-51133-7_21

are Tabu Search [4], Simulated Annealing [5], Genetic Algorithms [6, 7], and Ant Colony Optimization [8].

This paper presents a different methodology to solve JSSP using genetic algorithms. A genetic algorithm (GA) is as a class of metaheuristic techniques that is aimed at finding solutions based on the mechanisms of natural selection and genetics. From an initial population, each individual is evaluated by a fitness function which depends on the purpose. Crossover and mutation operators are used for new generation of individuals in order to ensure a better access to the search space.

The paper is organized as follows. The detailed presentation of the JSSP is provided in Sect. 2. Section 3 provides the experimental results to solve a set of deterministic JSSP in comparison with different works that have been currently referenced. Section 4 presents the conclusions and suggests some aspects that could be subject to further researches.

2 Proposed Methodology

The proposed methodology suggests a simple indirect representation based on a binary matrix and, therefore, the use of simple genetic operators, such as the one-point crossover and bit-flip mutation. Despite its simplicity, it proved powerful enough by its association with the dynamic seed, which is based on the inheritance of the previous seed that generated the best solution within a certain number of generations and its best individual, as further explained in the next sections.

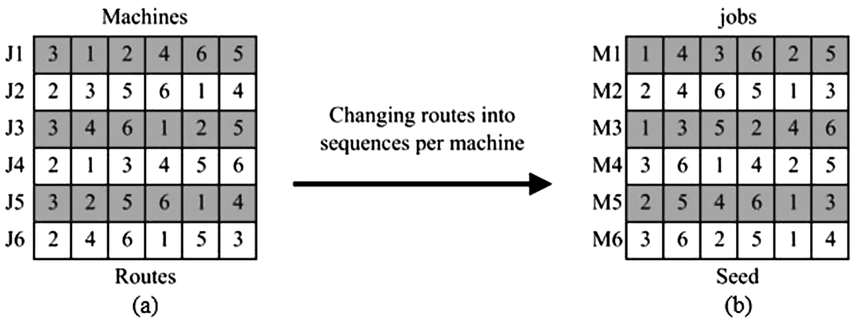
In order to evaluate the simulation code developed, 10 different sequences for the FT06 problem, generated from LISA [9], were presented to the simulation code. LISA is an open-source software developed by a group of researchers at Otto-von-Guericke University (under the supervision of Prof. Dr. Frank Werner) for creating, editing and troubleshooting deterministic scheduling problems, which has a useful graphical interface.

2.1 Representation Mechanism

The representation adopted in this work is based on a binary matrix of dimension $m \times (n - 1)$, where m is the number of the machines and n is the number of jobs. Since this representation does not allow a direct interpretation of a solution for the problem, Figs. 1 and 2 are used in order to clarify the process of decoding, considering a specific instance of JSSP known as FT06, available at the OR-Library [10].

First, the matrix of routes (Fig. 1a), which is given by the problem, is converted into a sequence of jobs per machine, according to the natural order of arrival of jobs, which is the seed for the solutions (Fig. 1b). To that end, the initial matrix is read by columns, from left to right, for each machine, in ascending order of jobs. This process is executed just once within GA, at the first run of the optimization process.

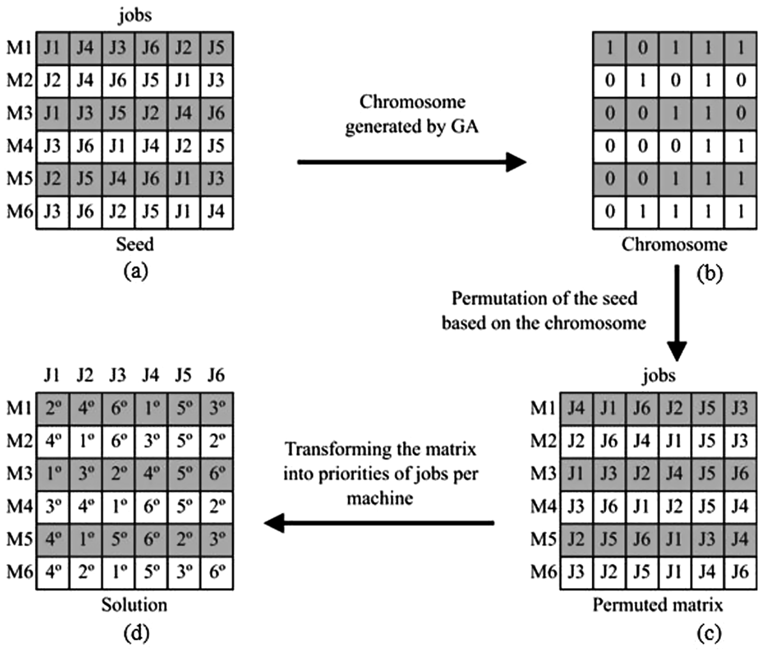
Thus, reading the first column in the matrix of routes, job 1 comes first in the machine 3, followed by jobs 3, 5, and jobs 2, 4 and 6 from the columns 2, 3 and 6 in the



J1	3	1	2	4	6	5
J2	2	3	5	6	1	4
J3	3	4	6	1	2	5
J4	2	1	3	4	5	6
J5	3	2	5	6	1	4
J6	2	4	6	1	5	3

M1	1	4	3	6	2	5
M2	2	4	6	5	1	3
M3	1	3	5	2	4	6
M4	3	6	1	4	2	5
M5	2	5	4	6	1	3
M6	3	6	2	5	1	4

Fig. 1. Generation of the seed (b) based on the routes (a)


jobs
Seed
(a)
Chromosome generated by GA
Chromosome
(b)
Permutation of the seed based on the chromosome
jobs
Permuted matrix
(c)
Transforming the matrix into priorities of jobs per machine
Solution
(d)

M1	J1	J4	J3	J6	J2	J5
M2	J2	J4	J6	J5	J1	J3
M3	J1	J3	J5	J2	J4	J6
M4	J3	J6	J1	J4	J2	J5
M5	J2	J5	J4	J6	J1	J3
M6	J3	J6	J2	J5	J1	J4

1	0	1	1	1
0	1	0	1	0
0	0	1	1	0
0	0	0	1	1
0	0	1	1	1
0	1	1	1	1

M1	J4	J1	J6	J2	J5	J3
M2	J2	J6	J4	J1	J5	J3
M3	J1	J3	J2	J4	J5	J6
M4	J3	J6	J1	J2	J5	J4
M5	J2	J5	J6	J1	J3	J4
M6	J3	J2	J5	J1	J4	J6

	J1	J2	J3	J4	J5	J6
M1	2°	4°	6°	1°	5°	3°
M2	4°	1°	6°	3°	5°	2°
M3	1°	3°	2°	4°	5°	6°
M4	3°	4°	1°	6°	5°	2°
M5	4°	1°	5°	6°	2°	3°
M6	4°	2°	1°	5°	3°	6°

Fig. 2. Representation schema of FT06 instance

matrix of routes, respectively; thereby generating the third row of the seed (Fig. 1b). Then, the process is repeated for all other machines.

The GA generates a binary matrix (Fig. 2b), which is the chromosome. After the first generation, the genetic operators act over this matrix. A reading of this matrix by rows is performed, from top to bottom, and where the value 1 is identified, a permutation in the seed is accomplished. The permutation occurs between the element in the same position where the value 1 in the chromosome was found, and its successor.

Therefore, as the value of first element of the chromosome (in Fig. 2b) is 1, a permutation between the values 1 and 4 in the seed occurs (Fig. 2a), generating the first element of the new array of sequences (Fig. 2c), which is the job 4, and the second element, that is the job 1. Until this moment, the job 1 is temporarily in the position 2, since his stay in that position depends upon the value of the second element in the chromosome.

As the value of the second position in the chromosome is 0, there will be not swapping between the second and third element of the original sequence. Thus, the first two elements of the new matrix are really the jobs 4 and 1, respectively. This process is repeated with all the elements of the first row of the seed, and then all the elements of the second row of the seed, and so forth. The reason to have just five elements in the rows of the chromosome is because the swapping occurs between the element in the current position and its successor, which means a value of 1 in the fifth position already generates permutation between the fifth and sixth elements of the original seed, and therefore is not necessary to have a sixth column (considering the problem FT06).

Finally, the new matrix generated through the permutation is turned into a matrix of priority of the jobs per machine (Fig. 2d), which is the way that the simulation code interprets the sequences. Analyzing the first three jobs in the first row of the permuted matrix is clear to see that job 4 should be scheduled first in the machine 1, followed by jobs 1 and 6, respectively. Therefore, in the matrix of the priorities (which is the representation of the solution itself), these are the jobs that receive priority values 1, 2, and 3, respectively. This process is conducted on all elements of the new matrix, called a solution (Fig. 2d).

2.2 Dynamic Seed Genetic Algorithm

The proposed approach, called Dynamic Seed Genetic Algorithm (DSGA), applies the classical GA as an inner level in which the candidate solutions are generated through permutations of the seed based on chromosomes, according to Fig. 2. Additionally, an external level is created in order to update the seed dynamically. After some generations of the classic GA in the inner level, a defined number of best solutions are chosen and a local search is performed in each of them, related to the original route of the problem addressed, thus generating new seeds. Specifically, the local search permutes the predecessor and successor of a chosen job that its delay will directly cause a delay that affects the whole system (i.e. a job that belongs to the so called critical path). Then, the best current solution and the corresponding permuted seed are used to update the seed from the previous step and a new set of generations in the inner level takes place.

3 Experiments

To prove the effectiveness of the proposed approach, a set of JSSP instances was taken from OR-Library, which is used by several researchers to compare their results against different methods and techniques. The instances set range from LA01 to LA10, due to Lawrence [11]. This set of problems deals with the classical deterministic job shop problem, which does not take into account the dynamic and stochastic behavior that can reflect more appropriately real-world industry situations. However, GA presents peculiar aspects in relation to other optimization methods and it is simple, flexible, robust and particularly useful in solving problems where other optimization techniques facing difficulties. Especially in relation to the fitness function, that may be a mathematical function, an experiment, a simulation model or a metamodel.

So, the proposed approach can be easily integrated with a simulation model in order to deal with a dynamic job shop scheduling problem that can take into account factors like, for example, random release and processing times, setup times, random machine breakdowns, and create more robust scheduling with regard to the stochastic and dynamically changes in the real manufacturing environment [12].

For each instance the proposed approach was run 10 times, in order to allow statistical references. The implementation was made through GAlib, which is a library of GA written in C++ by Matthew Wall, from the Massachusetts Institute of Technology [13]. Parameters adopted in GA are summarized in Table 1.

Table 1. Parameters of the GA adopted in the experiments

Parameter	Adopted value	Reason
Chromosome representation	Binary, matrix of dimension $m \times (n - 1)$	Proposed methodology
Selection	Steady state	[14]
Replacement rate	90%	Experiments
Population size	10	Experiments
Crossover	One-point	Proposed methodology
Crossover rate	10%	Experiments
Mutation	Bit-flip	Proposed methodology
Mutation rate	1%	Experiments
Total number of generations	20,000 (50 internal plus 400 external)	Experiments
Stop criteria	Number of generations	[15]
Fitness Function	Simulation code	Proposed methodology

3.1 Performance Related to the Quality of the Solution

Unlike to run the total number of iterations once with elitism, which is largely used in the classic GA, in addition to inherit the best individual, the proposed approach also copies to the next generations the generator seed that was associated with that better individual. So, the seed is updated and used to create new candidate solutions in the following inner level, which contributes to the convergence process.

This approach allows the GA to route new and possibly better paths, as the initial population after each external loop is based on the best seed from previous generations in the inner level (i.e., a classic GA). By route better paths we mean to create a larger number of feasible solutions during the evolution process of the genetic algorithm in relation to the classical elitism approach, as it can be seen in Table 2.

The results of the proposed approach were also compared to three different methodologies found in the current researches. The Memetic Algorithm approach

Table 2. Feasible solutions of LA01 problem: classic GA vs. DSGA

Classic GA (with elitism)			DSGA		
Run	Feasible solutions	Non-feasible solutions	Run	Feasible solutions	Non-feasible solutions
1	103377	21623	1	115866	9134
2	102310	22690	2	116068	8932
3	101788	23212	3	114116	10884
4	101450	23550	4	115649	9351
5	102594	22406	5	114525	10475
6	103293	21707	6	118684	6316
7	104061	20939	7	118174	6826
8	102746	22254	8	115275	9725
9	101706	23294	9	116231	8769
10	103665	21335	10	116218	8782
Average	82.16%	17.84%	Average	92.86%	7.14%

(MA) is due to [16], while the Differential Algorithm with Sub-Group (DE) is due to [17] and the Hybrid Genetic Algorithm (HGA) is due to [7].

In order to ease compare the approaches, Table 3 summarize the makespan values obtained from the proposed methodology and those reported from other recent methodologies used to compare with. It is important highlight that the approaches DE, MA and HGA are not based only on the AG and can therefore be even more difficult to program and use.

Table 3. Makespan values obtained from the different methodologies (BKS = Best Known Solution, Avg. = Average)

			Methodology							
			MA		DE		HGA		DSGA	
Instance	Size (*n* x *m*)	BKS	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
LA01	10 × 5	666	666	*NA*	666	666.0	666	*NA*	666	672.6
LA02	10 × 5	655	655	*NA*	655	663.6	655	*NA*	660	680.8
LA03	10 × 5	597	597	*NA*	597	610.4	597	*NA*	616	619.2
LA04	10 × 5	590	590	*NA*	590	597.3	590	*NA*	607	619.8
LA05	10 × 5	593	593	*NA*	593	593.0	593	*NA*	593	593.0
LA06	15 × 5	926	926	*NA*	926	926.0	926	*NA*	926	926.0
LA07	15 × 5	890	890	*NA*	890	890.0	890	*NA*	890	890.0
LA08	15 × 5	863	863	*NA*	863	863.0	863	*NA*	863	863.0
LA09	15 × 5	951	951	*NA*	951	951.0	951	*NA*	951	951.0
LA10	15 × 5	958	958	*NA*	958	958.0	958	*NA*	958	958.0

4 Conclusions and Suggestions

This paper presented a different methodology to solve Job Shop scheduling problems using Genetic Algorithms, which the authors called DSGA. The methodology works by using a simple indirect binary representation as the chromosome and simple genetic operators (one-point crossover and bit-flip mutation), and change the seed that generates a solution from time to time, initially defined by the original sequencing of the problem addressed, and then adopting the best individual from the past runs of the GA as the seed for the next runs.

The proposed methodology is easy to implement and, despite not finding the optimal solution for all instances, the simple proposed approach fails in problems in which the convergence to the optimal makespan is harder in all the different approaches, even for more sophisticated techniques which is especially true for square problems that are admittedly more difficult problems. Moreover, the proposed approach allows the GA to go through new and possibly better paths, which means to create a larger number of feasible solutions during the evolution process of the genetic algorithm in relation to the classical elitism approach.

As a continuity of this work, it is suggested to implement the proposed methodology in stochastic problems, with randomly processing times, obeying a probability distribution.

Acknowledgements. The authors would like to thank grant #2014/08688-4, São Paulo Research Foundation (FAPESP), for their funding of this research.

References

1. Pinedo, M.L.: Scheduling: Theory, Algorithms and Systems. Springer, New York (2008)
2. Jamili, A.: Robust job shop scheduling problem: mathematical models, exact and heuristic algorithms. Expert Syst. Appl. **55**(15), 341–350 (2016)
3. Braune, R., Zäpfel, G.: Shifting bottleneck scheduling for total weighted tardiness minimization—a computational evaluation of subproblem and reoptimization heuristics. Comput. Oper. Res. **66**, 130–140 (2016)
4. Bo, P., Zhipeng, L., Cheng, T.C.E.: A tabu search/path relinking algorithm to solve the job shop scheduling problem. Comput. Oper. Res. **53**, 154–164 (2015)
5. Faccio, M., Ries, J., Saggioro, N.: Simulated annealing approach to solve dual resource constrained job shop scheduling problems: layout impact analysis on solution quality. Int. J. Math. Oper. Res. **7**(6), 609–629 (2015)
6. Kurdi, M.: An effective New Island model genetic algorithm for job shop scheduling problem. Comput. Oper. Res. **67**, 132–142 (2016)
7. Qing-Dao-Er-Ji, R., Wang, Y.: A new hybrid genetic algorithm for job shop scheduling problem. Comput. Oper. Res. **39**(10), 2291–2299 (2012)
8. Huang, R., Yang, C.-L., Cheng, W.-C.: Flexible job shop scheduling with due window—a two-pheromone ant colony approach. Int. J. Prod. Econ. **141**(2), 685–697 (2013)
9. LISA: A Library of Scheduling Algorithms. http://www.math.ovgu.de/Lisa.html
10. Beasley, J.E.: OR-Library (2003). http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt

11. Lawrence, S.: Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Ph.D. dissertation, Carnegie-Mellon University (1984)
12. Menezes, F.M., Blanco, G.T., Rodriguez, P.C., Pereira, F.H.: Application of simulation and optimization for dynamic job shop scheduling under stochastic demand variability. In: Proceedings of 7th International Conference on Management of Computational and Collective Intelligence in Digital EcoSystems (MEDES 2015), vol. 1, pp. 1–8 (2015)
13. GAlib: A C++ Library of Genetic Algorithm Components. http://lancet.mit.edu/ga/dist/. Accessed 19 Nov 2014
14. Yamada, T.: Studies on metaheuristics for jobshop and flowshop scheduling problems. Ph.D. dissertation, Kyoto University (2003)
15. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
16. Gao, L., Zhang, G., Zhang, L., Li, X.: An efficient memetic algorithm for solving the job shop scheduling problem. Comput. Industr. Eng. **60**(4), 699–705 (2011)
17. Wisittipanich, W., Kachitvichyanukul, V.: Two enhanced differential evolution algorithms for job shop scheduling problems. Int. J. Prod. Res. **50**(10), 2757–2773 (2012)