A SIMPLE GENETIC ALGORITHM FOR THE JOB-SHOP SCHEDULING PROBLEM

LESTAN, Z.; BREZOCNIK, M.; BREZOVNIK, S.;
BUCHMEISTER, B. & BALIC, J.

Abstract: *The Job-shop scheduling is concerned with arranging processes and resources. Proper schedules are very important for the manufacturers, but can cause serious problems because of the enormous solution space. Pressure from the competitive enterprises is the main reason why time is becoming one of the most important success factors. The goal in this paper is the development of an algorithm for the Job-shop scheduling problem, which is based only on genetic algorithms. Our intention is to prove, that even a very simple genetic algorithm is capable for Job-shop scheduling. The effectiveness of the algorithm is demonstrated by solving practical problems.*

Key words: *job-shop scheduling, manufacturing, genetic algorithms, evolutionary computation*

Authors´ data: B.Sc. **Lestan** Z[oran]; D.Sc. **Brezocnik** M[iran]; B.Sc. **Brezovnik** S[imon]; D.Sc. **Buchmeister** B[orut]; D.Sc. **Balic** J[oze], University of Maribor; Faculty of Mechanical Engineering, Smetanova 17, SI – 2000 Maribor, Slovenia, zoki2905@gmail.com, mbrezocnik@uni-mb.si, simon.brezovnik@uni-mb.si, borut.buchmeister@uni-mb.si, joze.balic@uni-mb.si

This Publication has to be referred as: Lestan Z[oran]; Brezocnik M[iran]; Brezovnik S[imon]; Buchmeister B[orut] & Balic J[oze] (2010). A Simple Genetic Algorithm for the Job-Shop Scheduling Problem, Chapter 56 in DAAAM International Scientific Book 2010, pp. 645-654, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-901509-74-2, ISSN 1726-9687, Vienna, Austria
DOI: 10.2507/daaam.scibook.2010.56

1. Introduction

In the Job-shop problem a set of jobs must be processed on a set of machines. Each job consists of a sequence of operations, where each operation must be processed on a predefined machine for an exact time (Buchmeister et al., 2008). All operations of a job must be processed one after another in the given order. There are several constrains on jobs and machines (Pinedo, 2005):

- there are no precedence constrains among operations of different jobs,
- operations cannot be interrupted,
- each machine can process only one job at a time,
- a job does not visit the same machine twice,
- neither release times nor due dates are specified.

The objective is to minimize the makespan, i.e., the maximum of job completion times. In order to do this, the right operation sequences on the machines have to be determined.

Because the Job-shop scheduling problem is a very difficult NP problem, it has captured the interest of a significant number of scientists, so a lot of algorithms have been proposed for solving it (Brucker 2007). The algorithms which base on the method branch and bound are useful only for solving small instances. Instances with a larger number of operations cannot be solved in a reasonable time. For large scale problems approximation algorithms are used: shifting bottleneck, genetic algorithm, simulated annealing, tabo search, priority dispatch, etc. Although it is not possible to solve even modest sized instances to optimality, a number of important algorithmic advances have been made in the past few years (Mitsuo & Runwei, 1997). The latest studies indicate that the best results are obtained with the use of hybrid methods involving genetic algorithms, tabo search, simulated annealing and the shifting bottleneck approach. One of the most promising methods for Job-shop scheduling is a hybrid method based on simulated annealing and tabo search (Zhang et al., 2006). The advantage of tabo search is in a memory function which prevents that the search would end in a local optimum. If this method is combined with the method which is analogue to physical annealing and independent from the initial solution, a very effective search algorithm is made.

In this paper a simple genetic algorithm is used to treat the Job-shop problem. This algorithm was developed independently, without regard for the work of other researchers. The intention was to make a simple algorithm which will try to find the schedule with the smallest makespan. Only genetic operations are used in order to achieve this. It is possible to schedule a various number of jobs, but neither release times nor due dates are considered. Only selection and permutation are used as genetic operations. The algorithm uses random moves to search for the optimal schedule in the solution space, which means that the solution is obtained without the help of heuristic methods. This paper is organized as follows: in Section 2, the encoding method used in the algorithm is introduced. Section 3 describes the

evaluation of encoded solutions, in Section 4, the genetic operations are described, in Section 5, the influence of evolution parameters on the search procedure is explained, in Section 6, two experimental results are shown and in Section 7 is the conclusion.

2. Encoding

How to encode solutions to chromosomes to ensure feasible solutions is a key issue for genetic algorithms. Researchers use different representations for the Job-shop scheduling problem (Runwei & Mitsuo, 1996):

- operation-based representation,
- machine-based representation,
- disjunctive graph-based representation,
- preference list-based representation,
- job-based representation,
- priority rule-based representation.

In our algorithm, the preference list-based representation is used. In this encoding method the operations are arranged in a certain order. It depends on this order, how the operations will be processed on the machines. It is very important, that the precedence constrains of operations of individual jobs are considered. This means, that the sequence of operations of a job must stay intact also in the encoded solution. How the encoding works is shown on a simple example. Table 1 shows a 3x3 instance; 3 jobs (9 operations) must be scheduled on 3 machines (*M1, M2, M3*) to achieve the smallest possible makespan.

JOBS	PROCESSING TIMES			PROCESSING ORDER		
	operations			operations		
	1	2	3	1	2	3
J1	29	78	9	machine *M1*	machine *M2*	machine *M3*
J2	43	90	28	machine *M1*	machine *M3*	machine *M2*
J3	91	85	74	machine *M2*	machine *M1*	machine *M3*

Tab. 1. 3x3 instance

From the Table 1 it is possible to write the operation sequence for each job:

J1 (*J1 M1* 29) (*J1 M2* 78) (*J1 M3* 9)
J2 (*J2 M1* 43) (*J2 M3* 90) (*J2 M2* 28)
J3 (*J3 M2* 91) (*J3 M1* 85) (*J3 M3* 74)

Job *J1* must first be processed on machine *M1* for 29 time units. After that on machine *M2* for 78 units and the last is machine *M3* for 9 units. Similar goes for the other two jobs. The schedule for this instance is encoded into a string, where the position of the operation in the string plays an important role. Operations are ordered with the help of a randomizer. Therefore it is important to use a reliable randomizer. String making in our case looks like this:

1. A list of first operations of all jobs is made.

   ((*J1 M1* 29) (*J2 M1* 43) (*J3 M2* 91))

2. From the list of first operations, one operation is chosen randomly (let's say (*J2 M1* 43)). This operation is the first operation in the string. The operation is taken from the corresponding job and inserted into the string.

   J1 (*J1 M1* 29) (*J1 M2* 78) (*J1 M3* 9)
   J2 (*J2 M3* 90) (*J2 M2* 28)
   J3 (*J3 M2* 91) (*J3 M1* 85) (*J3 M3* 74)

   String:

   ((*J2 M1* 43))

3. Again a list of first operations of all the jobs is made and an operation is randomly selected from the list (let's say (*J1 M1* 29)). This operation is taken from the corresponding job and inserted as second operation in the string.

   J1 (*J1 M2* 78) (*J1 M3* 9)
   J2 (*J2 M3* 90) (*J2 M2* 28)
   J3 (*J3 M2* 91) (*J3 M1* 85) (*J3 M3* 74)

   String:

   ((*J2 M1* 43) (*J1 M1* 29))

4. The procedure is repeated until all the operations from the jobs are transferred into the string.

If the procedure, described above, would be continued until the end, the string could look like this:

String:

((*J2 M1* 43) (*J1 M1* 29) (*J1 M2* 78) (*J3 M2* 91) (*J2 M3* 90) (*J3 M1* 85) (*J2 M2* 28) (*J1 M3* 9) (*J3 M3* 74))

This string will be used later on for demonstrations. A closer look at the string reveals that the precedence constrains have been considered during the making of the string. Job operations in the string still have the same processing order, but now they are mixed together. Why this is so important is explained in the next section.

Because the string making is left to coincidence, it is possible to make a lot of versatile strings (organisms) which are necessary for the initial population.

3. Evaluation

Only feasible strings represent a solution to our problem and therefore it is very important that feasibility is maintained throughout the searching process. Because in our case the goal lies in the time optimization of schedules, we are interested in the makespan. Besides the makespan we are interested also in the processing order on the machines. For that reason, Gantt charts are used for string evaluation. On the y – axis of the Gantt chart the operations are applied and the x – axis represents the time. The Gantt chart (string evaluation) is done step by step with adding operations one after another. In our case the operations are added directly from the string, from the left side to the right side. The operations which are at the beginning of the string have a higher processing priority than those at the end. This means, that the Gantt chart and the makespan depend only upon the order in the string. That is why it is so important, that the operation order in the string is according to the precedence constrains. Otherwise the evaluation would give a false value. The Gantt chart for our string is shown on Figure 1.

String:

((*J2 M1* 43) (*J1 M1* 29) (*J1 M2* 78) (*J3 M2* 91) (*J2 M3* 90) (*J3 M1* 85) (*J2 M2* 28) (*J1 M3* 9) (*J3 M3* 74))

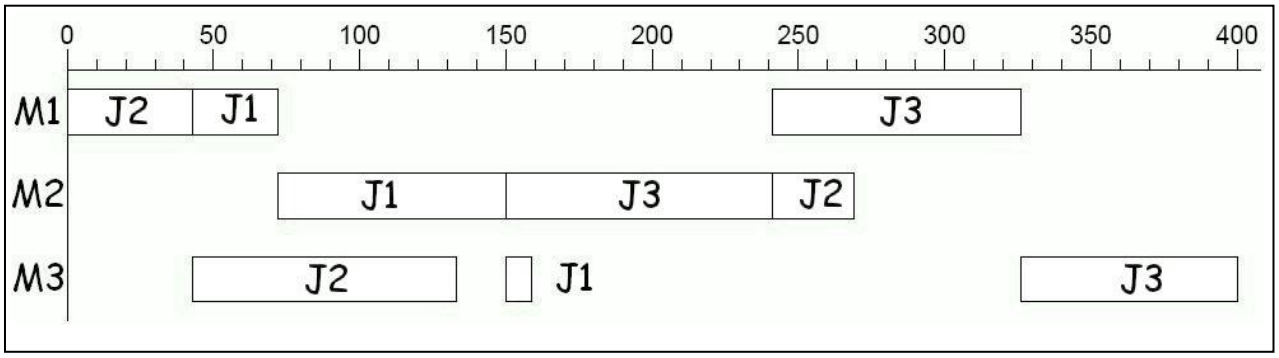

Fig. 1. Gantt chart for the 3x3 instance

Because the use of graphic Gantt charts in programming would be annoying, Gantt charts in numerical form are used. These charts are not as synoptic as graphical, but it is possible to comprehend all the important data from them. The Gantt chart on Figure 1 looks in numerical form like this:

M1 (0 *J2* 43) (43 *J1* 72) (241 *J3* 326)
M2 (72 *J1* 150) (150 *J3* 241) (241 *J2* 269)
M3 (43 *J2* 133) (150 *J1* 159) (326 *J3* 400)

4. Genetic operations

Genetic operations are the driving force in genetic algorithms. Which operations are reasonable to use for solving a certain problem depends on the encoding method. The only genetic operation, which is independent from encoding, is selection. Selection is the most simple of all genetic operations. In our algorithm the tournament selection is used. Its purpose is to maintain the core of good solutions intact. This is done with transferring good solutions from one generation to the next one. Because selection does not change the organism (string), we have no problem with maintaining feasibility, which is not the case in all other genetic operations. The use of the crossover operation can be problematical in some cases. The crossover operation often produces infeasible offspring, which are difficult to repair.

The only genetic operation besides selection, which is used in our algorithm, is permutation. The permutation is based on switching operations inside the organism. The organism which will be permutated is chosen with the selection. When executing the permutation it is necessary to consider the precedence constrains. The permutation procedure is described and shown on our string from Table 1:

1. A random operation is chosen from the string (let's say (*J2 M1* 43)).

   String:

   ((*J2 M1* 43) (*J1 M1* 29) (*J1 M2* 78) (*J3 M2* 91) (*J2 M3* 90) (*J3 M1* 85) (*J2 M2* 28) (*J1 M3* 9) (*J3 M3* 74))

2. The left and the right border for the chosen operation have to be defined. Because the chosen operation belongs to job *J2*, it is necessary to search for the first operation, left and right of the chosen operation, which belongs to job *J2*. If the operation does not exist, the border is represented by the end or the beginning of the string. In our case, the right border is represented by the operation (*J2 M3* 90) and the left border is presented by the beginning of the string. In the string, the space between the borders is marked with square brackets.

   String:

   ([(*J2 M1* 43) (*J1 M1* 29) (*J1 M2* 78) (*J3 M2* 91)] (*J2 M3* 90) (*J3 M1* 85) (*J2 M2* 28) (*J1 M3* 9) (*J3 M3* 74))

3. A random position between the brackets is chosen where the operation (*J2 M1* 43) is inserted (let us say in front of (*J1 M2* 78).

String:

((*J1 M1* 29) (*J2 M1* 43) (*J1 M2* 78) (*J3 M2* 91) (*J2 M3* 90) (*J3 M1* 85) (*J2 M2* 28) (*J1 M3* 9) (*J3 M3* 74))

This procedure randomly changes the chosen organism, which also changes the solution which the organism represents. Because there is often necessary to switch more than one operation in the organism, it is possible to repeat the whole procedure over and over.

5. Evolution parameters

Evolution parameters are parameters, which influence the searching procedure of the genetic algorithm. If we want to obtain good solutions, the parameters have to be set wisely. These parameters are (Mernik & Crepinsek, 2003):

- selection pressure,
- population size,
- amount of change, made by genetic operation,
- share of individual genetic operations in the next generation,
- number of generations (stopping criterion),
- number of independent civilizations.

The selection pressure defines what kind of solutions will be used in genetic operations. If the selection pressure is high, then only the best solutions will get the chance. If it is low, then also worse solutions are used. The higher the selection pressure, the higher the possibility that the search will end up in a local optimum. If it is too low, the search procedure examines insignificant solutions, which protracts the whole search.

When an organism is being modified, it is necessary to specify how much the genetic operation will change the organism. It is recommended that small and large changes are made to the organisms. This assures versatility in the population.

Each genetic operation must make a certain amount of organisms for the next generation. At least 10% of the next population has to be made with selection (Brezocnik 2000), so that the core of good solutions is maintained. All the other organisms are created with other genetic operations.

If the number of generations is multiplied with the size of the population, we get the number of organisms which have been examined during the search. When solving complex problems, the number of organisms is greater than in easier cases. The question is, should the search be executed with a small population and a lot of generations or opposite. In most cases a compromise is the best choice.

Because the search with genetic algorithms bases on random events, it is not necessary that good solutions are obtained in every civilization. In most cases the

search stops in a local optimum. That is why it is necessary to execute multiple searches for a given problem.

6. Results

The algorithm was tested on the notorious 10x10 and 20x5 instances which were proposed by Fisher and Thompson. Despite their small size, these two instances were very hard to solve. The 10x10 instance remained unsolved for nearly 25 years. Table 2 presents the 10x10 instance and Table 3 the 20x5 instance (Blazewicz 2007).

jobs	operation sequence									
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
J1	M1 29	M2 78	M3 9	M4 36	M5 49	M6 11	M7 62	M8 56	M9 44	M10 21
J2	M1 43	M3 90	M5 75	M10 11	M4 69	M2 28	M7 46	M6 46	M8 72	M9 30
J3	M2 91	M1 85	M4 39	M3 74	M9 90	M6 10	M8 12	M7 89	M10 45	M5 33
J4	M2 81	M3 95	M1 71	M5 99	M7 9	M9 52	M8 85	M4 98	M10 22	M6 43
J5	M3 14	M1 6	M2 22	M6 61	M4 26	M5 69	M9 21	M8 49	M10 72	M7 53
J6	M3 84	M2 2	M6 52	M4 95	M9 48	M10 72	M1 47	M7 65	M5 6	M8 25
J7	M2 46	M1 37	M4 61	M3 13	M7 32	M6 21	M10 32	M9 89	M8 30	M5 55
J8	M3 31	M1 86	M2 46	M6 74	M5 32	M7 88	M9 19	M10 48	M8 36	M4 79
J9	M1 76	M2 69	M4 76	M6 51	M3 85	M10 11	M7 40	M8 89	M5 26	M9 74
J10	M2 85	M1 13	M3 61	M7 7	M9 64	M10 76	M6 47	M4 52	M5 90	M8 45

Tab. 2. 10x10 instance

jobs	operation sequence				
	1.	2.	3.	4.	5.
J1	M1 29	M2 9	M3 49	M4 62	M5 44
J2	M1 43	M2 75	M4 69	M3 46	M5 72
J3	M2 91	M1 39	M3 90	M5 12	M4 45
J4	M2 81	M1 71	M5 9	M3 85	M4 22
J5	M3 14	M2 22	M1 26	M4 21	M5 72
J6	M3 84	M2 52	M5 48	M1 47	M4 6
J7	M2 46	M1 61	M3 32	M4 32	M5 30
J8	M3 31	M2 46	M1 32	M4 19	M5 36
J9	M1 76	M4 76	M3 85	M2 40	M5 26
J10	M2 85	M3 61	M1 64	M4 47	M5 90
J11	M2 78	M4 36	M1 11	M5 56	M3 21
J12	M3 90	M1 11	M2 28	M4 46	M5 30
J13	M1 85	M3 74	M2 10	M4 89	M5 33
J14	M3 95	M1 99	M2 52	M4 98	M5 43
J15	M1 6	M2 61	M5 69	M3 49	M4 53
J16	M2 2	M1 95	M4 72	M5 65	M3 25
J17	M1 37	M3 13	M2 21	M4 89	M5 55
J18	M1 86	M2 74	M5 88	M3 48	M4 79
J19	M2 69	M3 51	M1 11	M4 89	M5 74
J20	M1 13	M2 7	M3 76	M4 52	M5 45

Tab. 3. 20x5 instance

	10x10	20x5
population size	200	200
number of generations	500	500
number of independent civilizations	100	100
optimal solution	935	1165
best solution obtained	941	1211

Tab. 4. Results and evolution parameters for the 10x10 and 20x5 instances

As it can be seen from Table 4, the algorithm did not manage to find the optimal solution for the problem, but due its simplicity, the algorithm was able to obtain good solutions. Although both instances are made up of 100 operations, the solution space for the 20x5 instance is greater than that of the 10x10. The number of possible schedules *S* (solutions) can be calculated with equation (1), where *m* represents machines and *J* represents jobs.

$$S = (J!)^m \tag{1}$$

The search was in both cases executed with the same evolution parameters. Because the algorithm is written in the *LISP* programming language (Brezocnik, 2005) the search procedure took relatively long time (approx. 10 hours in both cases). In the 10x10 case, the best obtained solution deviates 1.2 % from the optimal solution. In the 20x5 case, the deviation was 4 %. The best result was is both cases obtained in only 1 civilization out of 100. The search procedure often falls in a local optimum. When this happens, it is very unlikely that the search will proceed to a better solution, because a memory function is not present. This means that the quality of the final solution depends on the initial population and pure chance. This is why the search procedure has to be repeated several times for a specific problem.

7. Conclusion

It is obvious that Job-shop scheduling plays an important role in the modern industry. The demanding market forces manufacturers to implement fast changes in their manufacturing space. These can be achieved only with the presence of a fast and powerful scheduler. Job-shop scheduling presents a hard combinatorial problem which is not easy to solve optimal. One of the methods for solving this stubborn problem is the genetic algorithm. Of course the performance of the algorithm depends on its design, which is linked with the experience and knowledge of the author. The truth is that not every evolution is a good evolution.

In this paper it is shown, that even a very simple evolution algorithm is capable of finding good solutions for the Job-shop problem. Because the genetic algorithms are not well suited for fine-tuning of solutions around optima the next step for this

algorithm would be an upgrade to a hybrid algorithm. The algorithm, presented in this paper, only imitates natural evolution to find better solutions. Like in nature there are no instructions on how to achieve this. With a little heuristic help, the search procedure would become more efficient. The easiest way to do this would be to involve the local search procedure into the existent algorithm. With this improvement only the operations on critical paths would be switched in permutation. This would definitely speed up the search process and the search would not stop in a local optimum as easy as before.

8. References

Blazewicz J., K. H. Ecker, E. Pesch, G. Schmidt, J. Weglarz: Handbook on sheduling, Springer-Verlag Berlin Heidelberg, 2007, ISBN: 978-3-540-28046-0

Brezocnik, M.: *Guide for the use of the autolisp tool in the AutoCAD environment (Prirocnik za uporabo orodja autolisp v okolju AutoCAD)*, Faculty of mechanical engineering, University in Maribor, 2005 (Fakulteta za strojnistvo, Univerza v Mariboru, 2005)

Brezocnik, M.: *The use of genetic programming in intelligent manufacturing systems (Uporaba genetskega programiranja v inteligentnih proizvodnih sistemih)*, Faculty of mechanical engineering, University in Maribor, 2000 (Fakulteta za strojnistvo, Univerza v Mariboru, 2000), ISBN: 86-435-0306-1

Brucker, P.: Scheduling algorithms: Fifth edition, Springer-Verlag Berlin Heidelberg, 2007, ISBN: 978-3-540-69515-8

Buchmeister, B., I. Palcic, J. Pavlinjek, INOKS d.o.o., Murska Sobota, University in Maribor Faculty of mechanical engineering : *Unconventional methods for Job-shop scheduling* (*Nekonvencionalne metode terminiranja proizvodnih procesov)*: Toolmaking 2008. Organizationt as driver for business improvements: consultation review, Portoroz, 7. – 9. October 2008 (Orodjarstvo 2008. Organizacija kot gonilo poslovnih izboljsav : zbornik posvetovanja, Portoroz, 7. do 9. oktober 2008.)

Mernik, M. Crepinsek, V. Zumer: *Evolutionary algorithms (Evolucijski algoritmi)*, Faculty of electrical engineering, computer and information, Institute for computer science, University in Maribor 2003, (Fakulteta za elektrotehniko, racunalnistvo in informatiko, Institut za racunalnistvo, Univerza v Mariboru, 2003), ISBN: 86-435-0593-5

Mitsuo G., Runwei Cheng: *Genetic Algorithms and engineering design*, John Wiley & Sons, Inc, New York, 1997, ISBN: 0-471-12741-8

Pinedo, M. L.: *Planning and scheduling in manufacturing and services*, Springer Science+Business Media, New York, 2005, ISBN: 0-387-22198-0

Runwei, C., Mitsuo, G., Yasuhiro, T., A tutorial survey of job-shop scheduling problems using genetic algorithms – I. representation, *Computers ind. Engng*, Vol. 30, No. 4, pp. 983 – 997, 1996

Zhang, C., PeiGen Li,YunQing Rao, ZaiLin Guan: *A very fast TS/SA algorithm for the job shop scheduling problem*, School of Mechanical Science & Engineering, Huazhong University of Science & Technology, Wuhan, 2006