ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2010, Vol.39, No.1

EFFICIENT GENERATION OF NEAR OPTIMAL INITIAL POPULATIONS TO ENHANCE GENETIC ALGORITHMS FOR JOB-SHOP SCHEDULING

Artur M. Kuczapski[1], Mihai V. Micea[1], Laurentiu A. Maniu[2], Vladimir I. Cretu[1]

[1] *Department of Computer and Software Engineering, "Politehnica" University of Timisoara, 2, V. Parvan Blvd., 300223, Timisoara, Romania*
artur.kuczapski@cs.upt.ro, mihai.micea@cs.upt.ro, vladimir.cretu@cs.upt.ro

[2] *Advanced clean production Information Technology (acp-IT), 7, Constantin Brancusi Str., 300050, Timisoara, Romania*
laurentiu.maniu@acp-it.com

Abstract. This paper presents an efficient method of enhancing genetic algorithms (GAs) for solving the Job-Shop Scheduling Problem (JSSP), by generating near optimal initial populations. Since the choice of the initial population has a high impact on the speed of the evolution and the quality of the final results, we focused on generating its individuals using genetically evolved priority dispatching rules. Our experiments show a significant increase in quality and speed of scheduling with GAs, and in some cases the evolved priority rules alone determined better solutions then the GA itself. The analyzed reference GA uses Giffler & Thompson (GT) heuristic and priority lists. To speed up the generation of priority rules, we have used a "weighted sum of priority rules" formula that revealed significantly better performances than Genetic Programming (GP). For evaluation of the proposed algorithm, the well known benchmark data sets from Fisher & Thompson (F&T) and Laurence Kramer (LA) have been used.

Keywords: genetic algorithms, job-shop scheduling, initial populations, chromosomes.

1. Introduction

Production scheduling is one of the hardest combinatorial optimization problems. A particular type of production scheduling is the Job-Shop Scheduling Problem (JSSP). This is the most addressed by researchers because it presents almost all the peculiarities of this domain and in the same time it has a quite simple formal representation. It was demonstrated that JSSPs are NP-hard and therefore no deterministic algorithms can solve them in a reasonable amount of time [1] – [3].

Genetic Algorithms (GAs) have proven their efficiency in solving high complexity combinatorial optimization problems, thus many researchers have applied them also to the scheduling problems [4] – [7]. The results seem to be encouraging but the quality of the schedules depends on several parameters. To improve the performance of the GA-based techniques, considerable effort was spent on developing optimal chromosome representation and genetic operations for JSSP [5], [7], [8]. However, the performance of solutions still depends on the quality of the initial population.

This paper describes an efficient method for generating near optimal initial populations in order to speed up GAs for JSSP. The proposed method also uses GAs, but in this case, for the purpose of evolving priority dispatching rules. These rules employ Giffler & Thompson (GT) heuristics to generate feasible schedules that serve as initial populations for the JSSP scheduling algorithms. To reduce the time required to find good dispatching rules, instead of Genetic Programming (GP) [9], [10], we use a "weighted sum of priority"-type of rules. This formula blends a set of predefined priority rules [2] – [4], [6], to create a better one.

To illustrate the performance of the proposed method, we use GAs for JSSP based on GT heuristic with ordinal representation of priority permutations [8]. The tests were run with the well known benchmark data sets from Fisher & Thompson [11] and Laurence Kramer [12], with and without Initial Population Generation (IPG).

2. Definition of the Job-Shop Scheduling Problem

A Job-Shop (JS) model contains a set of n jobs (J) and a set of m machines (M), where each job j visits a number of machines in a predetermined order. The processing times for each job at each machine are given and no machine can process more than one job at a time. If a job is started on a machine, then it cannot be interrupted (non preemptive), but any job can stay an arbitrary amount of time idle [1]. Briefly, the Job-Shop Scheduling Problem (JSSP) can be formulated as finding a job processing order at each machine that conforms to operations predetermined processing order of each job and minimizes or maximizes a given objective function [1][13].

To simplify, we further make the following assumptions which, nevertheless, are not detrimental to the generality of the proposed method:

- Each job has to undergo exactly m processing steps;
- No job can visit any machine more than once;
- The operation times are fixed and predetermined (no setup-times allowed);
- Any operation time is represented as a positive integer number which specifies the duration of the operation in terms of time units.

According to these statements, a JS can be described by a matrix of $n \times m$ entries JS[n, m], where each entry consists of two integer numbers. The row JS[j] describes the list of the operations, according to their execution order, for the job j, while the entry JS[j, i] specifies the machine needed for the operation i and the corresponding processing time.

There are several optimization objectives but the current paper will focus only on reducing the makespan (C_{max}), which is measured in time units, for all our case studies and tests.

3. Generating Dispatching Rules

3.1. Dispatching Rules

Many authors claim that priority dispatching rules can be successfully used in solving large JSSPs and even other scheduling problems [2], [3], [9]. In real applications, priority dispatching rules are actually the most widely used. Basically, a priority dispatching rule is a simple mathematical formula that, based on some processing parameters, specifies the priority of operations to be executed. The usual processing parameters are shown in Table 1.

There are several dispatching rules which present a significant optimization capacity [1], [3], [6]. Table 2 shows some of these rules, in unsigned format, where the higher or the lower value has the highest priority, depending on the JSSP. This representation has been chosen because it will directly support the evolution of dispatching rules.

Table 1. Production parameters for priority dispatching

Symbol	Description
r_j	Job arrival time – the moment when the job arrives at the machine
w_j	Job weight – the importance of the job
p_{jm}	Processing time – the time needed to process job j on machine m
n_j	Remaining operations – the number of remaining operations
R_j	Remaining work – the time needed to complete the job's remaining operations
P_j	Total work – the time needed to execute all the operations of the job
d_j	Due date – the moment when the job should be finished

Table 2. Simple priority rules used in the GA

Symbol	Expression	Description
AT	r_j	Arrival time (FIFO)
W	w_j	Weight
wPT	$w_j p_{jm}$	Weighted processing time
wWR	$w_j R_j$	Weighted remaining work
wTW	$w_j P_j$	Weighted total work
DD	d_j	Global due date
ODD	$r_j + (d_j - r_j)(R_j/P_j)$	Operation due date
$MODD$	$\max(ODD, t + p_{jm})$	Modified operation due date
ST	$d_j - R_j - t$	Slack time
$wSpO$	$w_i[1 - (d_j - R_j - t)/n_j)]/p_{jm}$	Weighted slack per operation
wCR	$w_i[1 - (d_j - t)/R_j)]/p_{jm}$	Weighted critical ratio

3.2. Evolving Dispatching Rules

Many researchers demonstrated that, for general JSSPs, simple priority rules alone cannot determine an optimal or near optimal solution [1], [3], [4], [9]. To improve the scheduling performance, they proposed the usage of evolutionary algorithms to generate better, composite dispatching rules (CDRs) which are mathematical combinations of various simple dispatching rules. Usually, Genetic Programming (GP) is employed for this purpose. The advantage of GP is that, theoretically, it can generate CDRs of any complexity, but on the other hand, generating high complexity CDRs requires a long computational time. To reduce the solution space and the computation time needed, the complexity of the CDRs was limited, and a smaller number of simple dispatching rules were selected.

To further increase the evolution speed, we opted for a CDR formula called "weighted sum of priority rules".

3.3. Chromosome Representation

To implement any evolutionary algorithm, first we need to define the representation of the solution. In our case the CDR is composed of a sum of priority rules ($PR_i(x)$) multiplied by their respective weights (pw_i):

$$CDR(x) = \sum_{i=1}^{N} pw_i \cdot PR_i(x), \quad (1)$$

where $CDR(x)$ is the composite dispatching rule generated; N is the number of priority dispatching rules used; pw_i is a real number in the interval [–1, 1] (the weight of the rule i); and $PR_i(x)$ is a priority dispatching rule from those defined in Table 2.

According to this representation the chromosomes that encode a CDR are constituted as a list of real numbers, from the interval [–1, 1], which has a length equal to the number of simple priority rules used. This representation has no additional constraints and it is similar to the classical chromosome representation, therefore all the genetic operations (crossover, selection, mutation) are well defined. In comparison with the GP approach, this representation has a much smaller solution space and therefore can evolve much faster, but in the same time, theoretically, it cannot find solutions as good as GP can.

Table 3. Genetic Programming compared to "weighted sum of priority rules" approach

Input data		OPT	GP		WSPR	
			C_{max}	Gn	C_{max}	Gn
FT6	(6x6)	55	58	58	55	23
FT10	(10x10)	930	1070	91	1043	27
FT20	(20x20)	1165	1264	77	1230	29
LA1	(10x5)	666	744	62	701	36
LA2	(10x5)	655	775	79	704	26
LA3	(10x5)	597	670	78	653	31
LA26	(20x10)	1218	1391	63	1348	35
LA27	(20x10)	1235	1547	66	1460	32
LA28	(20x10)	1216	1474	64	1460	35
LA29	(20x10)	1157	1462	83	1449	38
LA30	(20x10)	1355	1576	100	1560	35
LA36	(15x15)	1268	1546	51	1543	38
LA37	(15x15)	1397	1579	51	1580	29
LA38	(15x15)	1196	1415	70	1370	33
LA39	(15x15)	1233	1510	52	1417	34
LA40	(15x15)	1222	1356	96	1297	35

A set of tests have been conducted to compare the GP and the "weighted sum of priority rules" (WSPR) approaches. Table 3 shows the test results, where OPT stands for the best known (optimal) solution, C_{max} is the overall makespan, Gn is the number of generations needed to reach the solution, FT6 (6x6) is the 6x6 Fisher & Thompson problem (6 operations on 6 processing machines), LA1 (10x5) is the first Laurence Kramer problem (10 operations of 5 processing machines), and so on. The test results show that WSPR performs constantly better than GP.

3.4. Crossover

In our implementation we used the standard uniform crossover operation that, according to our tests, proved to be better than simple or multipoint crossover. Each crossover operation generates two complementary children whose genes are selected randomly from the two parents. The procedure of crossover is exemplified in Figure 1, where genes 2, 3 and 4 were randomly selected to be swapped.

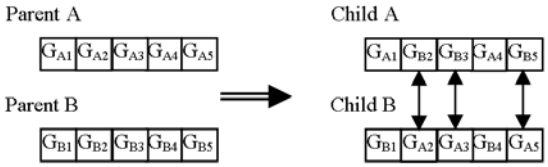

Figure 1. Uniform crossover operation example

3.5. Mutation

For mutation we have chosen to randomly alter a single gene in every chromosome selected to be mutated. The new value of the selected gene is generated randomly with a uniform distribution in the interval [–1, 1].

3.6. Reproduction

Although all genetic operations are well defined for canonical chromosome representation, we stress the fact that a significant performance increase can be obtained by using Stochastic Universal Sampling Selector (SUSS) [14] with exponential ranking.

Because usually the relative fitness difference between individuals in the same population is small, Roulette Wheel Selection proves to be insufficient to ensure convergence in the evolution [10]. To avoid this problem, we have used SUSS with linear and exponential ranking. The latter shows the best results with the following exponential formula:

$$P_i = \left(0.01^{1/k}\right)^{i-1}, \; i = \overline{1,k} \quad (2)$$

where P_i is the probability to reproduce the i-th individual, k is the number of individuals, and 0.01 (1%) is the probability to reproduce the worst individual.

3.7. Fitness Value

In order to use GAs, it is necessary to define a method that quantifies the "goodness" of a chromosome. In our case the fitness value of a chromosome is specified by the makespan (C_{max}) of the generated schedule. To generate the schedule we used GT heuristic [15]. In GT, when there are more operations that can be started on the same machine at the same

time, a decision rule is needed for selecting the first operation to run. In our case this decision rule is the CDR itself that we have to evaluate. When a schedule is generated the makespan is calculated.

3.8. Solutions Deviation

Figure 2 shows statistics for 100 consecutive runs of the CDR generation with GP and GA using WSPR. Both methods use the same population size, and stop condition. We can observe that even if GP occasionally generates better results, the GA-WSPR offers more reliable solutions with a better performance.

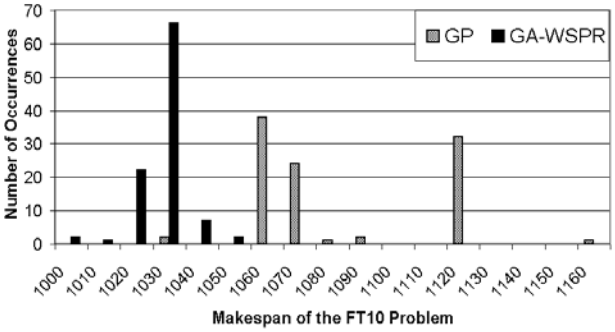

Figure 2. Distribution of 100 solutions of the FT10 dataset, using GP and GA-WSPR

4. Genetic Algorithms for JSSP

To evaluate the increase in performance, when using the proposed initial population generation method, we have implemented a simple genetic scheduler based on GT heuristic. In our implementation each individual in the population represents a list of priorities for each operation to be scheduled (permutation). To translate this priority list in schedules the GT heuristic is used.

4.1. Chromosome encoding

As each individual should represent a permutation, we have chosen the ordinary representation of permutation problems for GA [8]. Each chromosome consists of a list of n integer values, where n is the total number of operations in the JSSP, with the condition that each entry g_i of the list should contain an integer from 1 to $n - i + 1$. The meaning of each entry g_i is that on the i-th position of the permutation is the g_i-th element from a reference permutation (1, 2, ..., n). In other words, to translate a chromosome to a permutation one must iterate over the entries and for each g_i pick the g_i-th element from a reference list and add it to the i-th position of the permutation. When an element is picked from the reference list, it must be deleted, so that the number of available entries in the reference list is $n - i + 1$ for each iteration. Figure 3 illustrates the translation from ordinary representation to permutation.

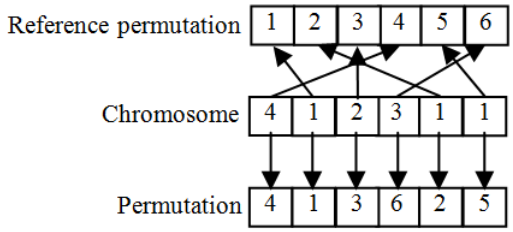

Figure 3. Translating ordinary representation to permutation

4.2. Genetic operations

This chromosome encoding has been used because the same genetic operation is applicable as in the case of GA-WSPR, so for crossover we have used the standard uniform representation, and for mutation, random gene altering with the condition that, if we alter the i-th gene of the chromosome then the new value should be generated for the interval $[1, n - i + 1]$.

5. Experimental Results

For evaluating the performance increase we have tested the presented GA scheduling with and without initial population generation (IPG) using GA-WSPR. The two algorithms were executed several times for each benchmark data set, and then the mean value was calculated. In addition, we have calculated the mean number of generations resulted upon the algorithm stop condition. The configuration used during the evaluation is presented in Table 4.

Table 4. Configuration of the test environment

Parameter	Value	
	CDR Generation	GA Scheduling
Population size	20	50
Crossover ratio	0.9	0.9
Mutation ratio	0.1	0.1
Random new individuals ratio	0.25	0.25
Number of populations to stop (stop criteria)	20	50

Table 5 synthesizes some of the most interesting measurement results. The input data column contains the same Fisher & Thompson and Laurence Kramer problems, as described in Table 3. In a similar way, OPT means the best known (optimal) solution, C_{max} is the overall makespan (measured in time units), Gn is the number of generations needed to reach the solution. The scheduling techniques compared are the First In First Out (FIFO) and the Genetic Algorithm (GA) – based schedulers. IPG stands for initial population generation.

Table 5. Experimental Results

Input data	OPT	FIFO	GA		IPG+GA		IPG	
	[*Gn*]	[C_{max}]	[C_{max}]	[*Gn*]	[C_{max}]	[*Gn*]	[C_{max}]	[*Gn*]
FT6 (6x6)	55	61	55	112	55	124	55	23
FT10 (10x10)	930	1228	1051	178	1007	187	1043	27
FT20 (20x20)	1165	1565	1295	200	1223	151	1230	29
LA1 (10x5)	666	772	676	154	668	195	701	36
LA2 (10x5)	655	899	697	26	677	149	704	26
LA3 (10x5)	597	771	628	141	640	179	653	31
LA26 (20x10)	1218	1433	1479	200	1316	166	1348	35
LA27 (20x10)	1235	1593	1556	186	1426	224	1460	32
LA28 (20x10)	1216	1557	1506	238	1403	206	1460	35
LA29 (20x10)	1157	1496	1481	208	1385	287	1449	38
LA30 (20x10)	1355	1614	1595	210	1492	269	1560	35
LA36 (15x15)	1268	1546	1500	191	1434	221	1543	38
LA37 (15x15)	1397	1579	1623	195	1554	194	1580	29
LA38 (15x15)	1196	1466	1442	185	1338	202	1370	33
LA39 (15x15)	1233	1532	1460	220	1397	177	1417	34
LA40 (15x15)	1222	1539	1438	96	1288	170	1297	35

6. Additional Scheduling Aspects

Although in this research we addressed just the JSSP, the proposed method can be used for almost any kind of scheduling problems.

We have also obtained good results in optimizing the Total Weighted Tardiness in Flexible Job-Shop with Recirculation (FJSR) scheduling problem. In this case the GT heuristic has been modified so that the additional peculiarities of this model could be handled.

7. Conclusions

In this paper we propose an efficient method of enhancing genetic algorithms for the Job-Shop Scheduling Problem by generating near optimal initial populations. We have shown that by using GAs and CDRs, it is possible to implement a fast evolutionary algorithm that generates good schedules in a short amount of time, which can be used as initial population for the target GA.

We have also shown that, for our purpose, it is much more efficient to use the "weighted sum of priority rules" representation than GP.

Finally we have proven that generating initial populations with CDRs can increase the quality of solutions up to 10% and, in the same time, the computational time can be reduced to up to 50%. Another advantage of the proposed method is that it can be easily adapted to other scheduling problems and it is feasible for large scale scheduling problems.

Acknowledgements

This work is supported by the Romanian Ministry of Education and Research, through the grant PNCDI II INOV-1262/2008-2011 and, in parts, through the grant PNCDI II ID-22/2007-2010.

References

[1] **M.L. Pinedo**. Planning and Scheduling in Manufacturing and Services. 1st Ed. *Springer New York*, 2006.

[2] **K. Kemppainen**. Priority Scheduling Revisited: Dominant Rules, Open Protocols, and Integrated Order Management. *Helsinki School of Economics*, 2005.

[3] **A.S. Jain, S. Meeran.** Deterministic job-shop scheduling: past, present and future. *European Journal of Operational Research*, 113 (2), 1999, 390–434.

[4] **J. Käschel, T. Teich, G. Köbernik, B. Meier**. Algorithms for the Job Shop Scheduling Problem: A Comparison of Different Methods. *European Symposium on Intelligent Techniques, Greece, Jun*. 1999.

[5] **C. Bierwirth**. A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms. *OR Spektrum*, 17, 1995, 87–92.

[6] **S.-C. Lin, E. D. Goodman, W. F. Punch, III**. A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems. *Proceedings of the Seventh International Conference on Genetic Algorithms*, 1997, 481–488.

[7] **J. Garen**. Multiobjective Job-Shop Scheduling With Genetic Algorithms Using a New Representation and Standard Uniform Crossover. *Workshop on Multiple Objective Metaheuristics, Paris, Nov*. 2002.

[8] **P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, S. Dizdarevic**. Genetic Algorithms for the traveling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review*, 13, 1999, 129–170.

[9] **N.B. Ho, J. C. Tay**. Evolving Dispatching Rules for solving the Flexible Job-Shop Problem. *IEEE Congress on Evolutionary Computation*, 3, 2005, 2848–2855.

[10] **W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone**. Genetic Programming: An Introduction – On the Automatic Evolution of Computer Programs and Its Applications. 1st Ed., *Morgan Kaufmann San Francisco*, 1997.

[11] **H. Fisher, G. L. Thompson**. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. *Industrial Scheduling, J. F. Muth and G. L. Thompson (eds.), Prentice-Hall Englewood Cliffs, NJ*, 1963, 225–251.

[12] **S. Lawrence**. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania*, 1984.

[13] **V. T'kindt, J.-C. Billaut**. Multicriteria Scheduling. *European Journal of Operational Research*, 167 (3), *Elsevier B. V., Dec*. 2006, 589-591.

[14] **J. E. Baker**. Reducing Bias and Inefficiency in the Selection Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms and their Application* (*Hillsdale*), 1987, 14–21.

[15] **B. Giffler, G. L. Thompson**. Algorithms for Solving Production-Scheduling Problems. *Operation Research*, 8 (4), 1960, 487–503.

[16] **T. Yamada, R. Nakano**. Genetic Algorithms for Job-Shop Scheduling Problems. *Proceedings of Modern Heuristic for Decision Support, UNICOM Seminar, London*, 1997, 67–81.

Received September 2009.

DOI: 10.5755/j01.itc.39.1.12091