Comparison of solution representations for scheduling in the unrelated machines environment

Marko Đurasević, Domagoj Jakobović
Faculty of Electrical Engineering and Computing, Zagreb, Croatia
Email: marko.durasevic@fer.hr, domagoj.jakobovic@fer.hr

Abstract – Most scheduling problems belong to the class of NP hard problems. Because of that reason, search based approaches are often used in order to find solutions for scheduling problems. In this paper we compare several search-based approaches for finding solutions for the unrelated machines scheduling problem. These search based approaches use two different solution representations for the aforementioned problem and the representations are compared with each other. The first representation uses a permutation vector to encode the solution, while the second one uses a vector of floating point numbers. The results of the search based approaches are compared to several existing heuristics developed especially for solving the unrelated machines scheduling problem. We also perform a complexity analysis of the search-based approaches, in which we compute the time needed for them to outperform the problem specific heuristics.

I. INTRODUCTION

Scheduling can be defined as a decision-making process concerned with the allocation of scarce resources to tasks over a given time in order to optimize one or more objectives [1], [2]. A great number of scheduling problems belong to the class of NP hard problems, meaning that no efficient algorithms exist which could find the optimal solution in a reasonable amount of time. Therefore, such problems are often solved by using different heuristic algorithms. Based on the nature of how the solutions are generated, heuristic algorithms are divided into two groups: the first group, denoted as *search-based*, consists of metaheuristic approaches which search the whole solution space in order to find the optimal solution, while the second group consists of problem specific heuristics which iteratively construct the solution [3].

Search-based approaches have the advantage that they are usually able to find near optimal solutions. However, this advantage comes at a price, since search-based approaches are rather slow when compared to approaches which build the schedule iteratively. Additionally, search-based approaches can mostly be used in scheduling environments where all information about the environment is available beforehand, since they search the whole solution space. In such cases, when all the information about the scheduling environment is present beforehand, these approaches are mostly the methods of choice, since they outperform simple constructive scheduling heuristics.

One of the most important design choices in search-based algorithms is the solution representation of the given problem. The overall effectiveness of search-based approaches can, to a great extent, depend on the chosen solution representation [4]. Most commonly a solution representation which can best represent the given problem is chosen, but that may not always be the best choice, since some other solution representations could produce better results. Because of that, this paper will compare two solution representations for scheduling in the unrelated machines environment. The first representation is a permutation representation similar to the ones used in the literature. The second representation uses a vector of floating point numbers in order to represent the schedule. To the best of our knowledge, the second representation has not yet been used to represent solutions for the unrelated parallel machines environment. The results of those two approaches will be compared to several iterative scheduling heuristics. Additionally, this paper will examine the time complexity of genetic algorithms (using the aforementioned solution representations) when compared to some traditional heuristic methods for creating schedules for the unrelated parallel machines environment.

The paper is organized as follows: in the second section a short overview of the unrelated machines environment is given. The third section describes the two solution representations in more detail. The results are presented in the fourth section, while a discussion about the results is given in the fifth section. Finally, a short conclusion and future research directions are presented in the sixth section.

II. RELATED WORK

A lot of work has been invested in solving different scheduling problems by using various metaheuristic methods [5], [6], [7], [8], [9], [10]. Unfortunately, very little research has been done in the area of the unrelated machines environment. In [11] the authors propose a hybrid algorithm combining ant colony optimization, simulated annealing and variable neighborhood search in order to solve scheduling problems for parallel unrelated machine environment. The ant colony optimization algorithm was also used in [12], where the authors propose several new ideas in order to improve the performance of the algorithm. A Greedy Randomized

Adaptive Search Procedure (GRASP) metaheuristic was proposed in [13]. A Competitive Evolutionary Strategy Memetic Algorithm which for optimizing two objectives simultaneously was proposed in [14].

Genetic algorithms have also been used to solve the unrelated machines problem on several occasions. For example, in [15], the author adapts a genetic algorithm for the unrelated machines environment and shows promising results. In [16] a genetic algorithm using a permutation representation and several local search operators was proposed. It was shown that the proposed method achieved excellent performance, when compared to other similar methods. GARP, a genetic algorithm combined with variable neighborhood descent and path relinking was proposed in [17]. In [18] a hybrid genetic algorithm was proposed for the unrelated machines environment with worker allocations. This article additionally examines several solution representations for the given problem. Solving the unrelated parallel machines environment as a subset of the flexible flow shop problem was discussed in [19].

III. Scheduling in the Unrelated Machines Environment

The unrelated machines environment consists of *n* jobs which compete in order to be processed on one of the *m* machines [2]. Each job is defined by several properties including the processing time p_{ij}, which defines the execution time of job with the index *j* on the machine with the index *i*; the release time r_j, which defines the time when the job arrives in the system; the due date d_j, which defines a point in time by which the job should finish with its execution, otherwise a certain cost will be caused by its delay; and a weight w_j, which denotes the importance of a job.

A. Scheduling Criteria

In order to assess the quality of a given schedule, certain scheduling criteria need to be defined. Since the number of the proposed scheduling criteria is numerous, four of the most prominent criteria from the literature are chosen. But before defining the criteria for the entire schedule, it is essential to first define certain criteria for individual jobs, which will then in turn be used to define the scheduling criteria.

For each job the following criteria are defined [1]:

- Finish time C_j – denotes the point in time in which the job ended with its execution
- Flowtime F_j – denotes the amount of time a certain job spent in the system before it finished with its execution, which is defined as: $F_j = C_j - r_j$,
- Tardiness T_j – denotes the amount of time that the job was executing after its due date. It is defined as: $T_j = \max(C_j - d_j, 0)$
- Is tardy U_j – denotes if a job is tardy or not. It is defined as: $U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}$

In this paper the following four scheduling criteria will be optimized:

- Makespan $C_{\max}$ – represents the maximum finishing time of all jobs. It is defined as: $C_{\max} = \max(C_j)$
- Total flowtime Ft – represents the sum of all job flowtimes. It is defined as: $Ft = \sum_j F_j$
- Total weighted tardiness Twt – represents the weighted sum of the tardiness values of all jobs, and is defined as: $Twt = \sum_j w_j T_j$
- Weighted number of tardy jobs Uwt – represents the weighted sum of tardy jobs. It is defined as: $Uwt = \sum_j w_j U_j$

B. Scheduling Conditions

Scheduling problems can be classified into several classes depending on some of their properties. For example, based on the availability of system parameters (for example, the information about all jobs which will arrive to the system) scheduling problems can be divided into *offline scheduling*, in which all information is available from the start, and *online scheduling*, in which the information about a job becomes available only when that concrete job enters the system). On the other hand, scheduling problems can also be divided based on when the schedule is created. In *static scheduling*, the entire schedule is created before the system begins with its execution, while in *dynamic scheduling* the schedule is built simultaneously with the execution of the system.

Scheduling conditions can have a great influence on the algorithms which can be used for creating schedules. For example, search-based approaches can mostly be used in static offline scheduling; on the other hand, methods which build the schedule iteratively are mostly used in dynamic online scheduling, but can be also used in static and offline scheduling.

In this paper we presume the static scheduling conditions, where all the parameters are known before the jobs enter the system.

C. Constructive Scheduling Heuristics

Constructive scheduling heuristics are simple iterative procedures which build the schedule incrementally. The main concept of all such heuristics is the same: each time a machine becomes available and there are jobs to be scheduled, the heuristic selects one of the jobs and schedules it on the available machine. Specific heuristics then differ from each other based on the rule which is used to select the job. This rule can, for example, select jobs in the following ways: the job with the shortest processing time, the job with the highest weight, the job with the closest due date, etc. For the unrelated machines scheduling environment, there exists a wide variety of different scheduling heuristics, but for this article, we will limit ourselves to the following four popular scheduling heuristics: min-min [20], max-min [21], sufferage [22] and min-max [23].

IV. Solution Representations in Genetic Algorithms for Scheduling Problems

A. Genetic Algorithms

Genetic algorithms are a stochastic optimization technique which can be used in order to find high quality solutions for a wide variety of problems [24], including for different scheduling problems. The main idea of genetic algorithms is to simulate the process of natural evolution in order to find good solutions. The algorithm starts with a randomly generated population of solutions, where each solution is called an individual. In order to compare individuals with each other, a fitness function needs to be defined. The role of the fitness function is to determine how "good" a certain individual is. Based on that information the *selection* operator chooses "better" individuals to survive, while trying to eliminate the "worse" individuals. Other operators, like *crossover* and *mutation* are used to explore the search space of the problem. Crossover usually tries to combine two individuals into a single individual which contains features of both parents, and which will hopefully represent a better solution. Mutation, on the other hand, introduces random changes into an individual, in order to extend the search to less covered regions. The algorithm iteratively applies the aforementioned genetic operators on a population, until a given stopping criteria is met.

Since each solution representation uses a different set of genetic operators, the actual list of used genetic operators will be given in the following subsections with the descriptions of the representations.

B. Permutation representation

The permutation representation is commonly used in order to represent solutions to scheduling problems. Throughout the literature many different variations of permutation representations have been used [4]. Which variant will be used, greatly depends on the scheduling environment and scheduling conditions of the problem.

In this paper, a representation consisting of two vectors will be used. The first vector will represent a permutation of jobs, which determines the sequence in which the jobs will be executed. But with this information alone it is not possible to determine on which machine a certain job should be executed. Because of that, a second vector of integers is used to denote on which machine a certain job will be executed.

9	4	3	7	2	0	8	1	5	6
0	2	2	1	0	0	1	0	2	1

Figure 1 Solution representation in the permutation based GA

Figure 1 represents an example of a solution representation for a scheduling problem with three machines and ten jobs. This solution represents a schedule in which the jobs with the indices 9, 2, 0 and 1 will be executed (in that order) on the machine with the index 0, jobs with the indices 7, 8 and 6 will be executed on the machine with the index 1, while jobs with indices 4, 3 and 5 will be executed on the machine with index 2.

Since this representation uses two vectors, where the first one represents a permutation, while the second represents just a normal array of integers, two different sets of genetic operators were used for each vector. For the first vector, 13 different crossover operators have been used, including OBX, OX, PBX, PMX and CX [10]. The operators are combined in a way that each time a crossover is performed, a randomly selected operator is applied.

For the second vector, only the uniform crossover operator was used. This operator takes two parent integer vectors and combines them into a single child vector, where at each position a value from a randomly selected parent is copied into the child.

Regarding the mutation operators, three have been used for the permutation vector (insert, inverse and swap mutation) and one for the integer vector (simple mutation). The simple mutation changes a value at the randomly selected position in the vector.

C. Floating point representation

The floating point representation, which usually consists of one vector holding floating point numbers, is most commonly used for solving continuous problems. Since scheduling problems are not continuous but combinatorial, a decoding scheme which will transform the floating point vector into a schedule needs to be devised.

The presented representation uses a floating point vector with size equal to the number of jobs. Each element in the vector is a number in the interval [0, 1]. These numbers denote priorities for each job, where 0 represents the highest priority, while 1 represents the smallest priority. This information is still not enough in order to create a complete schedule, since the mapping between jobs and machines is still undefined. For that reason, the interval [0, 1] is divided into m subintervals and

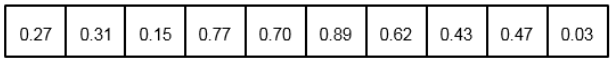

Figure 2 Solution representation in the floating point based GA

depending to which subinterval the priority value belongs, it will be mapped to the appropriate machine.

Figure 2 represents an example solution for a scheduling problem with three machines and ten jobs, encoded in the floating point representation. Since the scheduling problem contains three machines, the interval [0, 1] is divided into three subintervals: [0, 0.33> for the machine with index 0, [0.33, 0.66> for the machine with index 1 and finally [0.66, 1] for the machine with index 2. When a priority value belongs to a certain subinterval, it means that the job to which that priority value belongs will be mapped to the corresponding machine. In this example, the jobs with indices 0, 1, 2 and 9 will be mapped to the machine with index 0, jobs with indices 6, 7 and 8 will be mapped to the machine with index 1, and jobs with indices 3, 4 and 5 will be mapped to the machine with index 2. Since the mapping from each job to each machine has been resolved, only the sequences of those jobs on each machine need to be determined. For the first machine the job with index 9 has the smallest priority value, which means that this job will be scheduled first, following by jobs with indices 2, 0 and 1. The sequence for the other jobs is determined in the same way.

For this solution representation we used 15 different crossover operators, including BLX, arithmetic, one point and heuristic [25]. The crossover operators are combined in the same way as for the permutation encoding. As for the mutation, only the simple mutation operator is used, which changes the value of a randomly selected element of the vector.

D. Algorithms

Both the permutation and floating point encoding are used within a genetic algorithm that employs the same selection method. In this paper a steady state genetic algorithm is used with the tournament size of $k = 3$ individuals, which is given in Figure 3.

The floating point representation has an additional advantage that it can be used with existing continuous optimization algorithms other than GA. To illustrate that,

```
Steady-state tournament selection
{  randomly select k individuals;
   remove the worst of k selected;
   child = crossover(best two of k selected);
   perform mutation on child, with given
   individual mutation probability;
   insert child into population;
}
```

Figure 3. Pseudo-code of the steady state tournament GA

in this paper we also experiment with a memetic genetic algorithm, which incorporates a local search method. The search method in this case is based on pattern search Hooke-Jeeves deterministic algorithm [26]. The goal of incorporating a local search method is to guide the algorithm towards the nearest local optimum, in the hope it will also represent a globally optimal solution. This usually comes at a cost of a higher number of evaluations needed to reach the same objective value. The analysis of efficiency of different local search methods is out of the scope of this paper, but we include an example of this approach to demonstrate its applicability.

V. RESULTS

In this section, results for both solution representations will be presented. The genetic algorithm with the permutation solution representation will be denoted as GA-PERM, while the genetic algorithm with the floating point representation will be denoted as GA-FP. An additional memetic algorithm which uses Hooke-Jeeves local search, coupled with floating point encoding only, is denoted with GA-HJ.

A. Benchmark setup

In order to test the different representations, a benchmark set consisting of 60 scheduling problem instances was prepared. In order to test the algorithms on scheduling problems with different characteristics, the problem instances were designed with different number of jobs and machines. Depending on the specific problem instance, the number of machines can be 3, 6 or 10 and the number of jobs can be 12, 25, 50 or 100. More details about the design of the problem instances can be found on the project website [27]. In order for the benchmarks to be statistically significant, the algorithms were executed 30 times for each problem instance. The complete result for each algorithm is calculated as the sum of the best achieved solutions over all problem instances.

B. Parameter tuning

Since it is known that parameter values can have a significant influence on the obtained results, it is of great importance to perform a thorough optimization of the algorithm parameters, not only to obtain better results, but also in order to make the comparison as fair as possible and to determine the sensitivity of algorithms to the parameter values. First off, the maximum number of function evaluations has been set as the common termination criterion to allow for a fair comparison between different algorithms. The maximum number of function evaluations has been set to one million in order to give all algorithms enough time to converge. Two other parameters, population size and mutation probability, have been optimized for each algorithm separately. Both parameters were tuned by optimizing the *Twt* criterion, and then used for the other three criteria.

Table 1 represents the results achieved for various population sizes by the three genetic algorithm variants, with the best result bolded for each algorithm. The quality of solutions found by GA-PERM steadily increases with the increase of the population size, achieving best results

Table 1 Weighted tardiness optimization with different population sizes

Algorithm	Population size			
	30	100	200	1000
GA-PERM	11.14	10.42	10.15	**9.816**
GA-FP	**9.577**	9.850	9.70	9.617
GA-HJ	**17.68**	22.20	25.69	41.99

for a population size of 1000 individuals. The other two approaches show best results for the smallest population size of 30 individuals. While the solution quality degrades with the increase of population size for the GA-HJ algorithm, for the GA-FP algorithm, the solution quality drops for the population size of 100 individuals, but then for greater population sizes the solution quality slowly increases.

Table 2 represents the results for various mutation probability values. From the results it is apparent that all algorithms prefer higher mutation values. Both GA-PERM and GA-HJ achieve the best results for the highest mutation probability of 0.7, while GA-FP achieves the best results for a mutation probability of 0.5.

C. *Result comparison*

Table 3 shows the results for all the tested criteria achieved by the search based approaches, and additionally by four constructive scheduling heuristics selected from the literature: min-min, max-min, sufferage and min-max. The best values achieved by either the group of scheduling heuristics and search based approaches have been bolded. As it can be seen from the table, there does not exist a single scheduling heuristic which achieves supreme results for all criteria. The sufferage heuristic and min-min heuristic each achieve the best results for two criteria. The max-min heuristic, on the other hand, achieved the overall worst results for all criteria.

From the genetic algorithm approaches, GA-PERM and GA-FP achieved results which were better than results from all the scheduling heuristics. GA-HJ, on the other hand, was able to surpass the results of scheduling heuristics in only a few cases, and was generally not very effective. When the genetic algorithm approaches are compared with each other, GA-FP found the best solutions for all criteria, followed closely by GA-PERM. GA-HJ

Table 2 Weighted tardiness optimization with different mutation probabilities

Algorithm	Mutation probability				
	0.05	0.1	0.3	0.5	0.7
GA-PERM	9.837	9.920	9.816	9.773	**9.725**
GA-FP	9.767	9.673	9.577	**9.533**	9.566
GA-HJ	17.45	18.38	17.68	17.50	**16.77**

Table 3 Complete results for all scheduling criteria

Algorithm	Criteria			
	Twt	Nwt	Ft	Cmax
Constructive				
Min-min	16.71	**7.143**	**157.2**	38.31
Max-min	22.06	8.138	195.8	38.83
Min-max	17.49	7.793	167.3	38.06
Sufferage	**16.65**	7.194	160.9	**37.92**
Search-based				
GA-PERM	9.725	5.615	151.2	37.14
GA-FP	**9.533**	**5.340**	**140.8**	**36.79**
GA-HJ	16.77	6.395	232.1	41.67

came last with the significantly worst results from all three search-based approaches.

D. *Execution time comparison*

Apart from the obtained solution quality, the execution time very often represents an important factor in the choice of algorithms for solving a certain problem. Because of that, it is important to be aware of the time complexities of all the different approaches.

In order to compare the execution time of genetic algorithms with the constructive heuristics, we examine the quality level of the GA during its execution. When a solution is found by the GA which is of equal or better quality than that of a corresponding heuristic, we record the time at which that solution was obtained. The same process is repeated for 30 independent runs, and the resulting average time is used for runtime comparison.

Note that the GA is in general not guaranteed to converge to a better solution for every problem instance; in that case, a maximum runtime needed for one million evaluations is recorded. In our experiments, both GA-PERM and GA-FP were always able to find a better solution. The GA-HJ variant, however, was consistently worse and is therefore excluded from the comparison.

Table 4 represents the average execution times of the scheduling heuristics, GA-PERM and GA-FP. The first column represents which scheduling heuristic is used as a reference, the second column represents the criterion which was optimized, the third column represents the execution times of the scheduling heuristics, while the rest of the columns represent results for GA-PERM and GA-FP. It can be seen that the scheduling heuristics can construct the schedule for all 60 problem instances in around 0.2 seconds (depending on the heuristic). On the other hand, the time needed by the GA-PERM and GA-HJ algorithms to reach solutions of the same quality ranged from around 3 minutes up until 22 minutes, depending on the criterion. Regarding the algorithm, it can be seen that the GA-FP was able to find solutions of the necessary quality in less time than GA-PERM. Depending on the criteria, GA-PERM and GA-FP need more time to find the necessary solutions for the *Cmax* and *Ft* criteria, than for the *Twt* and *Nwt* criteria, but that behavior is expected since these scheduling heuristics are more suited to the

former two criteria (which can also be seen from the results).

VI. Discussion

Based on the results outlined in the previous section, several things can be concluded. First of all, it was shown that GA-FP achieved the best results for every scheduling criterion. GA-PERM came second, achieving results which are a few percent worse than those of GA-FP. This could come as a bit of a surprise, considering that the permutation representation is much more intuitive to represent solutions of scheduling problems. But on the other hand, the permutation representation needs two vectors to represent a complete solution, while the floating point representation encodes everything in a single vector. As a consequence, GA-PERM needs to perform genetic operators on both vectors, and it seems that this proves to be more challenging than performing genetic operators on only a single vector.

GA-HJ on the other hand achieved the worst results among the three GA approaches. This behavior was certainly not expected, since the Hooke-Jeeves local search was introduced to improve the results. In order to detect the cause of such behavior, results for each test instance of GA-HJ were compared to results of GA-PERM and it was noticed that both approaches achieve similar or the same results on smaller scheduling problems with 12 or 25 jobs. On the other hand, GA-HJ performs significantly worse on the larger problem instances with 50 and 100 jobs. This is probably the consequence of the local search algorithm, which in each iteration probes at least one additional point in every dimension and evaluates it, thus significantly increasing the function evaluation count. It could be possible that this algorithm would need much more evaluations in order to reach solutions of the same quality, but this is not an acceptable solution, since the maximum evaluation count is already large enough.

When compared to scheduling heuristics, GA-PERM and GA-FP have obtained better results, which is expected since they search the whole solution space in order to find the best schedule. The differences are more prominent for the *Twt* and *Nwt* criteria, since the constructive heuristics do not consider due dates of the jobs.

The execution time of the constructive heuristics is almost negligible, since they can create schedules for 60 problem instances in only 0.2 seconds. This characteristic makes these approaches applicable in dynamic and real world environments, where scheduling decisions need to be made very fast, and the algorithm needs to be able to quickly react to changes in the environment (such as the arrival of new jobs).

In order to find solutions of the same quality as those found by the scheduling heuristics, GA-PERM and GA-FP needed much more time, which ranged from 3 minutes up until 22 minutes, depending on the algorithm and optimization criterion. In all cases GA-PERM needed more time in order to find those solutions than the GA-FP algorithm. This behavior could also be the consequence of GA-PERM operating on two vectors, thus needing more time in order to reach good solutions. Additionally, since genetic algorithms are stochastic, there is no guarantee that they will be able to locate good solutions on the first try. As a consequence, these algorithms need to be executed several times in order to ascertain that good solutions have really been found.

Table 4 Execution time comparison between scheduling heuristics and genetic algorithms

Scheduling heuristic (SH)	Criterion	SH execution time (s)	GA-PERM execution time (s)	GA-FP execution time (s)	Ratio GA-PERM/SH	Ratio GA-FP/SH	Ratio GA-PERM/GA-FP
Sufferage	Cmax	0.197	1188.2	820	6031	4162	1.45
	Ft		1184.8	609.6	6014	3094	1.94
	Nwt		525.1	219.6	2665	1114	2.39
	Twt		265.2	200	1346	1015	1.33
Min-min	Cmax	0.097	1139.2	806.3	11744	8312	1.41
	Ft		1296.4	644.4	13364	6643	2.01
	Nwt		496.5	209.7	5118	2161	2.37
	Twt		264.6	203.5	2727	2097	1.30
Min-max	Cmax	0.215	1182.8	848.3	5501	3945	1.39
	Ft		1127.9	562.2	5246	2614	2.01
	Nwt		479.8	228.5	2231	1062	2.10
	Twt		259.6	206.8	1207	961	1.25
Max-min	Cmax	0.213	1057.6	728	4965	3417	1.45
	Ft		1002.8	458.8	4707	2153	2.19
	Nwt		416.6	200.5	1955	941	2.08
	Twt		237.7	176.7	1115	829	1.35

VII. CONCLUSION

This paper compares two different solution representations for scheduling problems in the unrelated machines environment. Although both representations were able to obtain good results, the floating point representation achieved the best results for all criteria. Unfortunately, contrary to our initial expectations, by adding a local search operator for the floating point representation, we were not able to improve the results of the genetic algorithm approach. Additionally, the genetic algorithm approaches were compared to several constructive scheduling heuristics, and it was shown that they are able to find much better solutions than scheduling heuristics, but with substantially longer execution time. This just shows that there is a tradeoff between solution quality and the time needed to find those solutions. The choice of the appropriate approach will then depend on the scheduling environment and user requirements.

For future research it is planned to further investigate some other solution representations, and compare them to the representations analyzed in this paper. Additionally, other local search mechanisms will also be tried out for both solution representations.

REFERENCES

[1] M. P. Pinedo, "Scheduling: theory, algorithms, and systems," Springer Science \& Business Media, 2012.

[2] J. Y-T. Leung, "Handbook of Scheduling: Algorthms, Models, and Performance Analysis", Chapman & Hall/CRC, 2004.

[3] A. Jones, L. Rabelo, "Survey of job shop scheduling techniques," tech rep., NISTIR, National Institute of Standards and Technology.

[4] E. Hart, P. Ross, D. Corne, "Evolutionary Scheduling: A Review," Genetic Programming and Evolvable Machines, 6, pp. 191–220, 2005.

[5] H. Zhou, Y. Feng, L. Han, "The hybrid heuristic genetic algorithm for job shop scheduling," Computers and Industrial Engineering, pp. 191-200, 2001.

[6] L. Wang, H. Siegel, V. Roychowdhury, A. Maciejewsky, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," Journal of Parallel and Distributed Computing, pp. 8-22, 1997.

[7] H. Zhou, W. Cheung, L. Leung, "Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm,", pp. 637-649, 2009.

[8] W. Cheung, H. Zhou, "Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times", Annals of Operations Research, pp. 65-81, 2001.

[9] S. Petrovic, E. Castro, "A Genetic Algorithm for Radiotherapy Pre-treatment Scheduling", Applications of Evolutionary Computation, pp. 462-471, 2010.

[10] F. Werner, "Genetic Algorithms for Shop Scheduling Problems: a Survey".

[11] J. Behnamian, M. Zandieh, S. M. T. Fatemi Ghomi, "Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm", Expert Systems with Application, 9637-9644, 2009

[12] C. W. Lin, Y. K. Lin, H. T. Hsieh, "Ant colony optimization for unrelated parallel machine scheduling", International Journal of Advanced Manufacturing, 35-45, 2013.

[13] J. P. C. M. Nogueira, J. E. C. Arroyo, H. M. M. Villadiego, L. B. Goncalves, "Hybrid GRASP Heuristics to Solve an Unrelated Parallel Machine Scheduling Problem with Earliness and Tardiness Penalties", Electronic Notes in Theoretical Computer Science, 53-72, 2014.

[14] C. C. Chyu, W. S. Chang, "A Competitive Evolution Strategy Memetic Algorithm for Unrelated Parallel Machine Scheduling to Minimize Total Weighted Tardiness and Flow Time", Computers and Industrial Engineering, 1-6, 2010.

[15] S. Balin, "Non-identical parallel machine scheduling using genetic algorithm", Expert Systems with Applications, 6814-6821, 2011.

[16] E. Vallada, R. Ruiz, "A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times", 612-622, 2011.

[17] M. N. Haddad, I. M. Coelho, L. S. Ochi, M. J. F. Souza, H. G. Santos, "GARP: A New Genetic Algorithm for the Unrelated Parallel Machine Scheduling Problem with Setup Times", International Conference of the Chilean Computer Science Society, 2012.

[18] A. Costa, F. Cappadonna, S. Fichera, "A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times", International Journal of Advanced Manufacturing Technology, pp. 2799-2817, 2013.

[19] H. C. Chang, H. T. Tsai, T. K. Liu, "Application of Genetic Algorithm to Optimize Unrelated Parallel Machines of Flexible Job-Shop Scheduling Problem", International Conference on Control & Automation, 2014.

[20] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," Journal of Parallel and Distributed computing, pp. 810-837, 2001.

[21] E. Davis, J. M. Jaffe, "Algorithms for scheduling tasks on unrelated processors," Journal of the ACM, pp. 721-736, 1981.

[22] D. Hensgen, R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," Journal of Distributed Computing.

[23] H. Izakian, A. Abraham, V. Snasel, "Comparison of heuristics for scheduling independent tasks on heterogeneous distributed Environments," Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on, pp. 8–12, 2009.

[24] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolutionary Programs", Springer-Verlag, Berlin, 1992.

[25] S. Picek, D. Jakobović, "From fitness landscape to crossover operator choice," Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 815-822, 2014.

[26] R. Hooke, T. A. Jeeves, ""Direct search" Solution of Numerical an Statistical Problems," Journal of the ACM, pp. 212-229, 1961.

[27] D. Jakobović, "Test case design," http://gp.zemris.fer.hr/scheduling/Test_cases_design.pdf