Application of Genetic Algorithms and Visual Simulation in a Real-Case Production Optimization

DAVORIN KOFJAČ, MIROLJUB KLJAJIĆ
University of Maribor, Faculty of Organizational Sciences
Kidričeva cesta 55a, SI-4000 Kranj
SLOVENIA
{ davorin.kofjac, miroljub.kljajic }@fov.uni-mb.si http://kibernetika.fov.uni-mb.si

Abstract: - This paper presents a real case customized flexible furniture production optimization. Such a make-to-order production must be flexible to meet the customer's needs, which are changing frequently. Hence, a frequent review of the production process is needed to ensure near-optimal production schedule to meet the minimal makespan constraint. In such a case we are confronted with a tradeoff between makespan and optimization runtime to ensure efficient real-time scheduling. The genetic algorithm production scheduling optimization is presented to solve a job shop scheduling problem with recirculation. Several initial population generators, selection methods, and crossover and mutation methods are discussed and tested. The visual model of the furniture production process was developed during the research. Such a model is closer to end-user perception and is used to clarify the results of numerical optimization. It also enables the implementation of end-user's expert knowledge into the optimizer to reduce the GA search space with a goal of reducing the runtime to a minimum.

Key-Words: Production scheduling, Job-shop, Recirculation, Genetic algorithms, Simulation, Optimization

1 Introduction

Scheduling is one of the most important issues in the planning and operation of manufacturing systems. In the classical job shop scheduling (JSS) problem, n jobs are processed to completion on m machines. Each job consists of k operations, one per machine, with known processing times and distinct technological ordering, and each machine is continuously available from time zero, processing one operation at a time without pre-emption. The operations are to be sequenced so as to minimize makespan. The introduction of recirculation complicates the already difficult JSS problem that is NP-hard. Hence, heuristics or artificial intelligence (AI) techniques seem to be unavoidable for JSS problems [2]. Several techniques to solve complex JSS problems are described in [4]. In our preliminary study, genetic algorithms (GA) were used to optimize the production schedule. They are known to search efficiently in a large search space, without explicitly requiring additional information (such as convexity, or availability of derivative information) about the objective function to be optimized [13]. For this reason, in the last decade, they have been applied to many (combinatorial) problems, including scheduling, yet only a few researches have been conducted to solve a JSS problem with recirculation using GA or the similar CLONALG (clonal selection-based algorithm) [10] [14] [15].

The production schedule produced by GA is a numerical solution that the end-users find hard to understand, especially as they are usually not interested in the methodology itself. Hence, the optimization results must be presented visually by a Gantt chart or some other visual technique. Therefore, a visual simulation model was developed that represents the furniture production process. Results produced by GA are imported into the visual simulation model that runs the simulation based on those results. Visual representation of optimization results is closer to the end-user's perception and also enables the validation of the GA optimizer. Hence, many inconsistencies can be corrected. Furthermore, optimized schedule produced by GA is valid, but it may not be logical or close to end-user comprehension of the scheduling problem. Therefore, visual simulation model provides a way where users can see many illogical schedule sequences that GA are producing, thus enabling end-users and the researchers to incorporate as much knowledge as possible into the optimization procedure. Hence, the problem search space is reduced to achieve real-time solutions as confirmed by Choi and Yang [3]. Reduction of a search space is crucial since GA is a multi-point search algorithm that requires a lot of computational time [20].

2 Methodology

2.1 Problem formulation

The case study is dealing with a make-to-order furniture production, where furniture is produced in very small series or no series at all. Such a production demands a lot of flexibility since there are no big series where scheduling is easier to perform. Generally, each customer's order is different in our case. Therefore, production scheduling has to be performed often to ensure the optimal/near-optimal production schedule with the objective to minimize makespan.

The base for the scheduling is the bill of materials (BOM) needed to complete a product, e.g. a wardrobe. In this case, BOM would include a frame, drawers, a door etc. On the other hand, a drawer's BOM would include, for example, four sides, one bottom, sliders, and so on. Such a BOM can be represented by a tree structure as shown in Fig. 1.

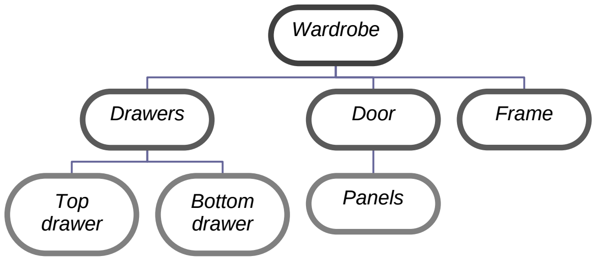


Fig. 1. A part of a wardrobe BOM represented by a tree structure.

One can easily notice that certain parts have to be completed before assembling of other parts can begin (precedence). Such a tree structure is basically a set of n jobs $J = \{ j_1, j_2, \ldots, j_n \}$ that have to be performed on a given set of m machines $M = \{ m_1, m_2, \ldots, m_m \}$. A job j_i consists of a set of k operations $O_i = \{ o_{i1}, o_{i2}, \ldots, o_{ik} \}$ that are performed on a subset of machines $H \subseteq M$. An operation o_{ik} is defined by the uninterrupted time period t_{ik} needed to perform this operation and a machine h_{ik} on which the operation is performed. In our case, a job can be repeated on the same machine (recirculation). Furthermore, jobs have to be processed in a given sequence as shown in an example presented by a tree structure in Fig. 1. The objective in our case is to produce a product in the shortest time possible, i.e. to minimize the makespan denoted by C_{max} that is defined as the time when the last job leaves the system:

$$C_{\max} = \max(C_1, C_2, \ldots C_n), \qquad (1)$$

where C_i is the completion time of job j_i.

2.2 Solving job-shop scheduling problem by genetic algorithms

The problem defined in the previous section can be described as the job-shop scheduling problem that is one of the best-known machine scheduling problems. A schedule is an allocation of the operations to time intervals on the machines. The problem is to find a schedule of minimum length [9]. The problem becomes even more complex if recirculation is introduced.

2.2.1 GA representation of the JSS problem

Lately, different GA representations for job-shop scheduling problems have been proposed, e.g. operation-based, job-based, machine-based, and so on [9]. During our research, we have used operation-based representation that encodes a schedule as a sequence of operations, and each gene stands for one operation. All operations for a job are named with the same symbol and are then interpreted accordingly to the order of occurrence in the sequence for a given chromosome, as proposed by Gen, Tsujimura and Kubota [9].

Consider the three-job four-machine problem given in Table 1. Jobs j_1 and j_2 have to be completed before the processing of job j_3 takes place. Hence, a chromosome is divided into sections (see Fig. 2), and crossover and mutation can be performed only inside a section.

Table 1. An example of a three-job four-machine problem - operation machine sequence for each job.

	Operations			
Job	1	2	3	4
j_1	m_1	m_3	-	-
j_2	m_1	m_3	m_4	-
j_3	m_1	m_2	m_1	m_4

Suppose a chromosome is given as [2 2 1 1 2 3 3 3 3], where 1 stands for job j_1, 2 for job j_2, and 3 for job j_3. Job j_1 has two operations (there are two 1's in the chromosome), job j_2 has three operations, and job j_3 has four operations. For example, the first 2 corresponds to the first operation of job j_2 which will be processed on machine m_1, the second 2 corresponds to the second operation of job j_2 which will be processed on machine m_3, and the third 2 corresponds to the third operation of job j_2 which will be processed on machine m_2 as shown in Fig. 3 (respectively for jobs j_1 and j_3). Note that operations 1 and 3 of the job j_3 are performed on machine m_1 (recirculation).

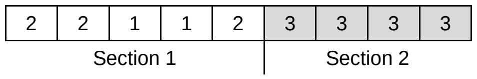


Fig. 2. A chromosome division into sections regarding the job precedence constraint.

2.2.2 Initial population

Initial population of chromosomes was generated in two ways: randomly and by mirroring. Random generation is a commonly used technique to provide initial population of chromosomes in the absence of a priori information about the solution. In the first step, *N* equal individuals are created as described earlier. Next, the genes in each section are moved to the random position inside the same section thus providing a random schedule.

Because random generation does not necessarily provide a good initial population, the mirroring of initial population was tested, inspired by the opposite-based population initialization proposed by Rahnamayan et al. [16]. They have proved that we can improve our chance of starting with a fitter solution by simultaneously checking the opposite solution. By doing this, the fitter one (random or opposite) can be chosen as an initial solution. In fact, according to probability theory, 50% of the time a guess is further from the solution than its opposite guess. Therefore, starting with the closer of the two guesses (as judged by its fitness) has the potential to accelerate convergence.

Mirroring is performed as follows. In the first step, initial population of size *N* is generated randomly. Next, each individual in randomly generated population is mirrored, i.e. each section inside an individual is mirrored, thus obtaining another *N* individuals. Individuals of both populations are then grouped and ranked according to the fitness function. Finally, best *N*-ranked individuals are taken as the initial population. By utilizing mirrored points, we can obtain fitter initial population even when there is no a priori knowledge about the solution(s). Here, the term "mirroring" is used intentionally instead of "opposite-based" because proving whether the mirrored schedule is actually an opposite-based schedule is exceeding the scope of this research.

Chromosome:	2	2	1	1	2	3	3	3	3
Machine:			1	2					

(a)

Chromosome:	2	2	1	1	2	3	3	3	3
Machine:	1	2			4				

(b)

Chromosome:	2	2	1	1	2	3	3	3	3
Machine:						1	2	1	4

(c)

Fig. 3. Operations of jobs and corresponding machines: (a) for job j_1, (b) for job j_2 and (c) for job j_3.

2.2.3 Selection

Several selection methods to choose chromosomes for the next generation were tested: Roulette selection (see [9]), Fitness rank distribution [17], and Fibonacci selection [1]. Also, the elite selection is performed to preserve the best individuals.

The Roulette selection is a fundamental selection method used in GA. In Roulette selection, the fitness function assigns a fitness value to chromosomes. This fitness value is used to associate a probability of selection with each individual chromosome. If f_i is the fitness of an individual *i* in the population, its probability of being selected is:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2}$$

where *N* is the number of individuals in the population. While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be.

The Fitness rank distribution (FRD) was introduced by Reeves [17]. It selects parents according to the following probability distribution:

$$p_i = \frac{2i}{M(M+1)} \tag{3}$$

where *i* refers to the *i*-th chromosome in descending order of makespan and *M* refers to the fittest one. This implies that the median value has a chance of 1/*M* of being selected, while the fittest one (the *M*-th chromosome) has a chance of 2/(*M* + 1), roughly twice the median.

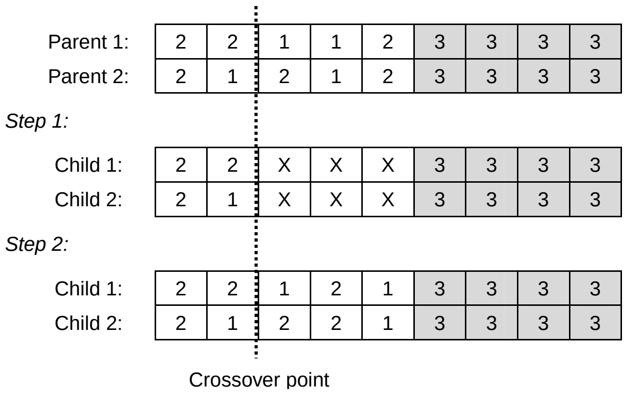
The Fibonacci selection was first introduced by Bernik [1] and uses Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, etc.) to select individuals. Fibonacci sequence is defined as follows:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n > 1. \end{cases} \tag{4}$$

Individuals from a population are selected according to the places defined by Fibonacci numbers. By applying this method, the fittest individuals are selected more often than the ones less fit. Hence, the variety of population is maintained and survival of the fittest concept is being applied, while the GA converges faster towards the solution.

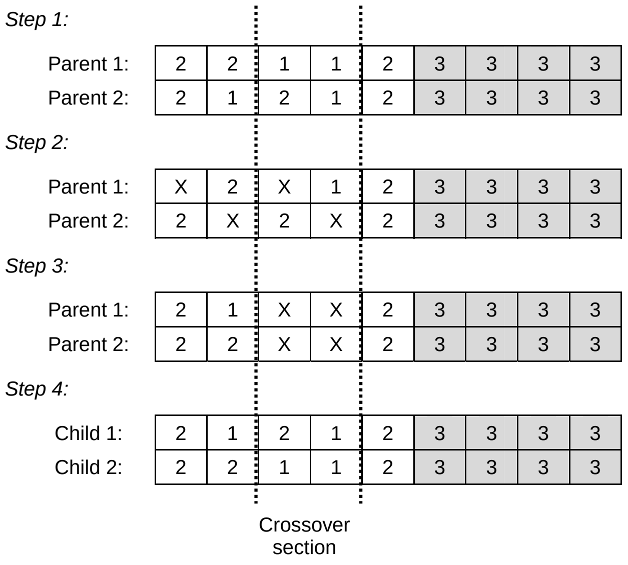
2.2.4 Crossover

To produce a new generation of chromosomes, a crossover is applied with a given probability p_c. The one-cut-point crossover (CUT) and linear order crossover (LOX) were tested. Note that crossover can be applied only inside a section (e.g. Section 1).



Parent 1: 2 2 1 1 2 3 3 3 3
Parent 2: 2 1 2 1 2 3 3 3 3
Step 1:
Child 1: 2 2 X X X 3 3 3 3
Child 2: 2 1 X X X 3 3 3 3
Step 2:
Child 1: 2 2 1 2 1 3 3 3 3
Child 2: 2 1 2 2 1 3 3 3 3
Crossover point

Fig. 4. An example of the CUT crossover in Section 1.



Step 1:
Parent 1: 2 2 1 1 2 3 3 3 3
Parent 2: 2 1 2 1 2 3 3 3 3
Step 2:
Parent 1: X 2 X 1 2 3 3 3 3
Parent 2: 2 X 2 X 2 3 3 3 3
Step 3:
Parent 1: 2 1 X X 2 3 3 3 3
Parent 2: 2 2 X X 2 3 3 3 3
Step 4:
Child 1: 2 1 2 1 2 3 3 3 3
Child 2: 2 2 1 1 2 3 3 3 3
Crossover section

Fig. 5. An example of the LOX in Section 1.

Consider two parents and two child chromosomes for CUT crossover as shown in Fig. 4. In the first step, genes [2 2] are copied from Parent 1 to the Child 1 up to the crossover point. Respectively, genes [2 1] are copied from Parent 2 to the Child 2. Child 1 is missing the genes [1 1 2] and the Child 2 is missing the genes [2 1 2]. The missing genes gap is filled in step 2. To produce a feasible schedule, the gap in each child must be filled with the missing genes by taking in order each legitimate gene from the other parent. For example, a gap in Child 1 is filled with the sequence [1 2 1], while the gap in Child 2 is filled with the sequence [2 2 1]. Note that the number of 1's and 2's in both "fill" sequences is equal to the one in the missing genes sequence, only the order in which 1's and 2's appear is different.

2.2.5 Mutation

After applying crossover, two types of mutation were tested: exchange (EX) and shift (SH) mutation. Mutations are applied to a child with a varying probability p_m (see [17]). The mutation rate is calculated as:

$$p_m = 1 - \frac{fit_{best}}{fit_{child}} \tag{5}$$

where fit_{best} represents the fitness function value of the chromosome that has yielded the best result and fit_{child} the fitness function value of the child that requires mutation. The chromosome with the fitness function value closer to the fit_{best} would have a lower p_m than the one with the fitness function value closer to the worst value.

The exchange mutation is performed by swapping the places of genes (chosen randomly) to produce a feasible solution (see Fig. 6). The shift mutation performs a shift of one gene (chosen randomly) to the right or left a random number of places (see Fig. 7). Again, the mutation can only be performed inside a section.

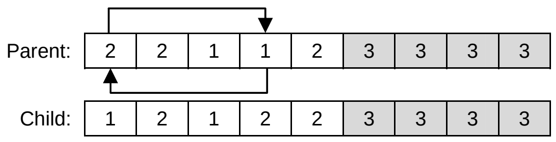


Fig. 6. An example of EX mutation in Section 1.

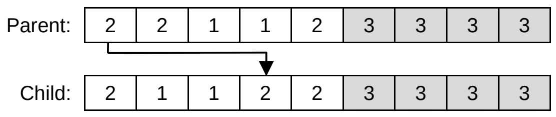


Fig. 7. An example of SH mutation in Section 1.

2.2.6 Population size

The effects of setting the parameters of evolutionary algorithms (EA) has been the subject of extensive research by the EA community and recently there is much attention paid to self-calibrating EAs that can adjust their parameters on-the fly (see e.g., [5] [6] for review). The most attention and most publications have been devoted to the adjustment of parameters of variation operators. Adjusting population size is much less popular, even though there are biological and experimental arguments to expect that this would be rewarding. Looking at it technically, population size is the most flexible parameter in natural systems: It can be adjusted much more easily than, for instance, mutation rate. In evolutionary computing, however, population size is traditionally a rigid parameter [7]. The new population resizing mechanism used here was introduced by Eiben et al. [7] and is based on improvements of the best fitness in the population. On fitness improvement the algorithm becomes more biased towards exploration increasing the population size, short term lack of improvement makes the population smaller, but stagnation over a longer period causes populations to grow again. Technically, this approach applies three kinds of changes in the population size:

1. If the best fitness in the population increases, the population size is increased proportionally to the improvement and the number of evaluations left until the maximum allowed. The formula used for calculating the growth rate GR_1 for is:

$$GR_1 = incFact \cdot (maxEvalNum - currEvalNum) \cdot \cdot \frac{maxFitness_{new} - maxFitness_{old}}{initMaxFitness} \tag{6}$$

   where *incFact* is an external parameter from the interval (0,1), *maxEvalNum* and *currEvalNum* denote the given maximum number of fitness evaluations and the current evaluation number, $maxFitness_{new}$, $maxFitness_{old}$ and *initMaxFitness* are the best fitness values in the current generation, the same in the preceding generation and the best fitness value in the initial population. (Note that we assume the existence of *maxEvalNum*, which is very often present indeed. In case it is not given, a very large number can be used instead.)
2. The population size is increased by a factor GR_2 if there is no improvement during the last V number of evaluations. In principle, the mechanism to increase the population size in this step can be defined independently from the previous one, but in fact we use the same growth rate, i.e. GR_1.
3. If neither 1. nor 2. was executed, then the population size is decreased. For the decrease rate DR a little percentage of the current population size is used, e.g. (1-5%).

In addition, we have added a war or disease process in the population growth process, similar to the one introduced by Shi et al. [17]. It is known, that natural environment does carry some capacity that cannot be exceeded; hence we shrink the population to its initial size after the population has exceeded some user defined maximum population size.

2.3 Visual simulation model of the flexible furniture manufactory

Computer-based simulation is seen as an integral business tool, giving flexibility and convenience to designing, planning and analyzing complex processes and/or systems. This is because computer-based modeling and simulation methods have the capability of representing the complex static structure as well as the dynamic behavior of systems [12][17]. Clearly, the imaginative and disciplined application of dynamic modeling and simulation provides a potentially useful mechanism through which users can gain a comprehensive understanding of system behavior [5].

Visual simulation model is used for validation of GA optimization algorithm that may produce a valid production schedule, yet this schedule may not be logical from the end-user's perspective. Through visual simulation users can notice illogical sequences that can be eliminated in GA optimization by applying some constraints. Visual model also allows incorporating as many knowledge as possible from experts (end-users) into the optimization algorithm, thus reducing the problem search space [3]. Hence, optimization algorithm is able to find an optimal/near-optimal solution much faster or even in real-time. On the other hand, visual simulation model can provide users with information on how to improve the production process itself. Users can run different simulation scenarios to see how the production process would perform if some parameters would have been changed (what-if analysis). Similar conclusions were drawn by [11], who have also used GA to integrate process planning and scheduling in a job shop.

Visual simulation model is presented in Fig. 8, representing only a fraction of the whole furniture production. It was implemented using Flexsim software for discrete-event simulation (DES). The software enables 3D visual simulation, import of simulation parameters, and export and representation of simulation results. The simulation model consists of machines used in the actual production process. Input into the model are the production schedule as a result of GA optimization, and setup and process times for each operation on a given machine that are extracted from a database.

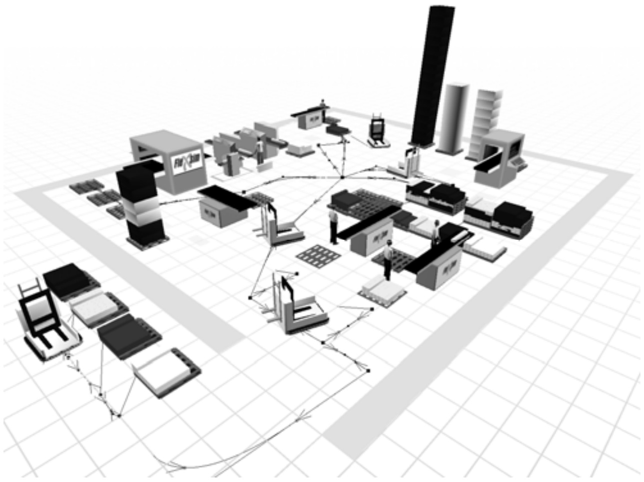


Fig. 8. A part of the furniture production as modeled with Flexsim simulation software

3 Results

The experiment was performed on a computer with Intel Core2 Quad 2.4GHz processor and 4GB RAM. The GA optimization algorithm was implemented with MS Visual Studio 2008 in C# technology.

The number of machines needed to perform JSS is 21. Three JSSP configurations were tested during the experiment:

1. BOM1 – 96 jobs divided into 4 sections with a chromosome length of 447 genes.
2. BOM2 – 101 jobs divided into 4 sections with a chromosome length of 558 genes.
3. BOM12 – a combination of BOM1 and BOM2.

Fig. 9 represents the distribution of jobs regarding the number of operations per job for BOM1 and BOM2. The job distribution for BOM1 is presented with a darker column and for BOM2 with a brighter one. Most jobs in both cases require six operations, while the minimum is one and maximum is 13 operations per job. Jobs with no operations are considered as assembly jobs that do not require any operation to be performed but are needed to satisfy the precedence constraint. Approximately 20% of jobs do not require recirculation.

3.1 Initial population

The first experiment was to determine whether mirroring of initial population yields fitter individuals. The experiment was performed with a population size of 50 individuals where 100 initial populations were generated for each case. First, the average makespan of each population was calculated. Second, the average makespan of all populations was calculated for each case. The results are shown in Fig. 10, where the average makespan of random and mirrored initial population regarding BOM1, BOM2 and BOM12 is compared. One can notice that mirroring yields a fitter initial population in all cases, thus reducing the average initial makespan for 37 mins (4,13%) in case of BOM1, 45 mins (3,95%) in case of BOM2, and 66 mins (3,74%) in case of BOM12. Applying mirroring to generate the initial population does not significantly slow the optimization runtime because it is executed only once at the beginning of the optimization process; on the other hand, it provides a better starting point for optimization.

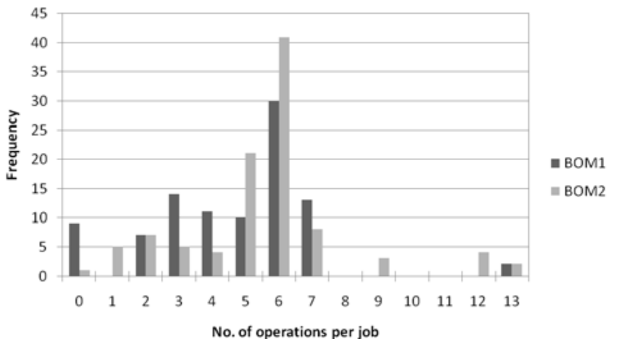


Fig. 9. A job distribution regarding number of operations per job for BOM1 and BOM2.

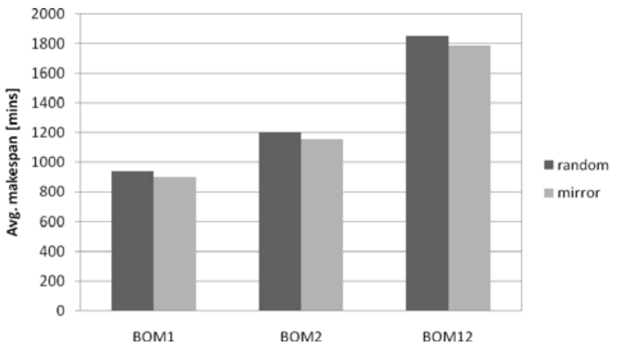


Fig. 10. The average initial makespan of random and mirrored initial population regarding each BOM.

3.2 Selection

The second experiment was performed to determine which selection method performs best in our case. Ten optimization runs were performed for each BOM. For each run, the optimization was performed until there was no makespan improvement for ten consecutive generations. Population size in this experiment was 100, with an EX variable mutation rate, CUT rate of 60%, and LOX rate of 40%. Fig. 11 represents the total makespan and runtime regarding the selection method. The makespan is presented as the total time to complete BOM1, BOM2, and BOM12; respectively for the runtime. The makespan results are shown with a solid line, and runtime results with a dashed one.

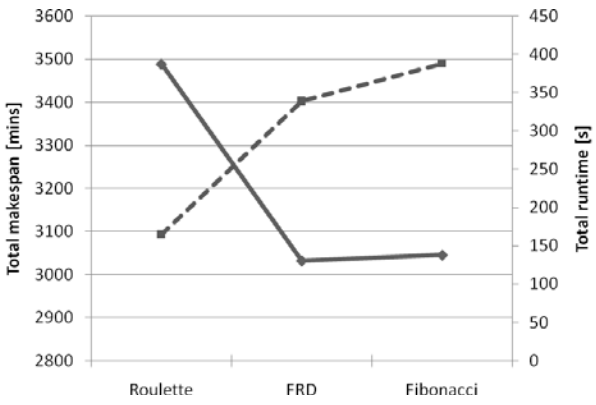


Fig. 11. The total makespan and runtime results regarding selection methods.

The Roulette selection has achieved the longest makespan (3489 mins) while the other selection methods have achieved similar results. Between the latter, the FRD selection has slightly outperformed Fibonacci selection (3030 vs. 3045 mins).

Optimization runtime is significantly affected by selection methods as shown in Fig. 11, where the total optimization runtime is presented. The Roulette selection has yielded shortest runtime (165s), yet it has also achieved significantly worst makespan results. The FRD selection has produced shorter runtime than Fibonacci selection (339 vs. 388s).

We can conclude that Roulette selection provides the shortest runtime, yet it also provides the worst makespan that is significantly outperformed by FRD and Fibonacci selection. The results show that Roulette selection does not provide a population that would converge towards a global optimal solution in a satisfying way - it obviously gets stuck in some of the local optimums. Therefore, its execution is stopped very quickly, since there is no improvement from the local optimum. The FRD and Fibonacci selection obviously provide a population that converges significantly better towards the global optimum. Hence, the runtime is much longer than in case of Roulette selection, because more generations are needed to reach global optimum. For further experiments we have chosen the FRD selection which has provided better makespan and runtime results than Fibonacci selection in most aspects.

3.3 Crossover and mutation

The third experiment was performed to determine which crossover and mutation methods perform best with FRD selection that was chosen as a result of the second experiment. Ten optimization runs were performed for each BOM and for each combination of methods. For each run, the optimization was performed until there was no makespan improvement for ten consecutive generations. The Population size in this experiment was 100, with an EX variable mutation rate, CUT rate of 60%, and LOX rate of 40%. The comparison of applying different crossover and mutation methods is presented in Fig. 12. The makespan results are presented with a solid line, and runtime results with a dashed one. The makespan is presented as the total time to complete BOM1, BOM2, and BOM12; respectively for the runtime.

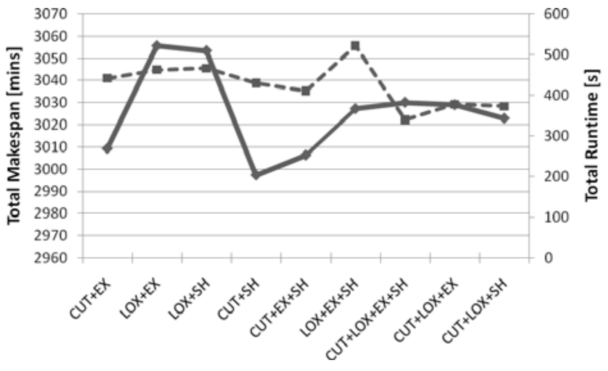


Fig. 12. The comparison of different crossover and mutation methods regarding the makespan and runtime.

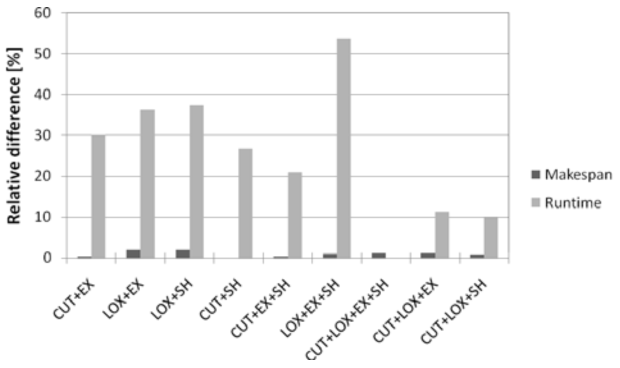


Fig. 13. The comparison of relative difference between the best combination of methods regarding the makespan and runtime.

The shortest total makespan was achieved by a combination of CUT+SH methods (2997 mins); the runtime was 430s. The longest makespan was achieved by a combination of LOX+EX methods (3055 mins); the runtime was 462s. The difference between the longest and shortest makespan is 58 mins (1,95%). Also worth mentioning is the total makespan of a combination of CUT+LOX+EX+SH methods (3030 mins) that is 1,1% slower than the shortest makespan. The shortest runtime (339s) was produced by a combination of

CUT+LOX+EX+SH methods, while the longest (522s) was produced by a combination of LOX+EX+SH methods. The difference between shortest and longest runtime is 183s (54%). Now, we will go into details about these differences.

Fig. 13 represents the relative difference regarding the best total makespan and runtime. One can notice small differences (with a maximum deviation of 1,95% regarding the best result of CUT+SH methods) regarding makespan by using different crossover and mutation methods. On the other hand, there are significant differences while observing runtime with a maximum deviation of 54% regarding the best runtime result of a CUT+LOX+EX+SH combination of methods. The results clearly show that applying different combinations of methods does not significantly improve the makespan. On the other hand, it does significantly affect the runtime. Clearly, by applying CUT+LOX+EX+SH methods we achieve 1,1% worse result regarding makespan as if we would by applying the CUT+SH methods, yet the runtime is significantly shorter (26%). In this case we would opt for a combination of CUT+LOX+EX+SH methods because there is bigger benefit of runtime than makespan.

3.4 Population size

The fourth experiment was to determine the population size to use with FRD selection and a combination of EX+SH+CUT+LOX methods that have yielded the best makespan results (Fig. 14). Ten optimization runs were performed for each BOM and for each combination of methods. For each run, the optimization was performed until there was no makespan improvement for ten consecutive generations. In this experiment we have used the EX variable mutation rate, CUT rate of 60%, and LOX rate of 40%. Again, the makespan is represented by a solid line and runtime with a dashed one.

Clearly, larger population produces a shorter makespan but also a longer runtime. The longest makespan (3052 mins) was produced by the smallest population of 50 individuals; in this case, runtime was 177s. On the other hand, the shortest makespan (3007 mins) was produced by the largest population of 150 individuals; in this case, runtime was 554s. The timespan between the shortest and longest makespan is 45 mins, a difference that cannot be neglected. On the other hand, to achieve the shortest makespan, one would need 554s instead of only 177s; again, a difference that cannot be neglected. Yet we would opt for the middle solution where a population of 100 individuals has been tested (makespan 3030 mins; runtime 358s) because of the tradeoff between makespan and runtime; we try to achieve a good makespan result while maintaining an acceptable runtime to ensure real-time scheduling.

The next experiment regarding population size was performed with a variable population size as described in Section 2.2.6. The test configuration was the same as with fixed population size and the results are shown in Fig. 15. The makespan results are presented with a solid line, and runtime results with a dashed one.

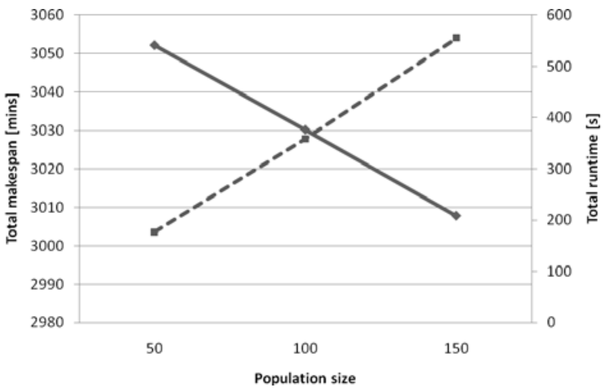


Fig. 14. The comparison of results regarding the fixed population size of the GA with FRD selection and a combination of EX+SH+CUT+LOX.

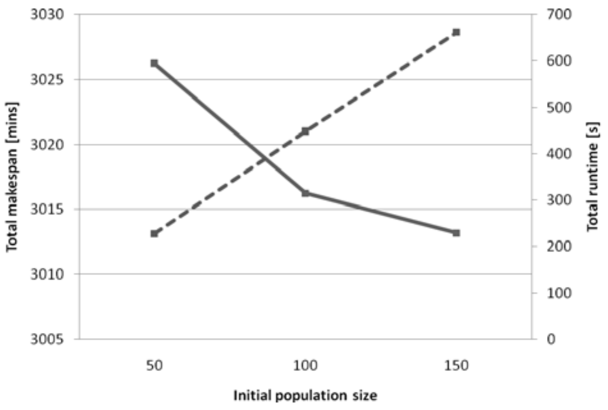


Fig. 15. The comparison of results regarding the variable population size of the GA with FRD selection and a combination of EX+SH+CUT+LOX.

The longest runtime (3026) was achieved by initial population of 50 individuals; the runtime was 228s. The shortest makespan (3013 mins) was achieved by initial population of 150 individuals, yet the runtime here is considerably longer (661s). The difference between the shortest and longest makespan is not as prominent with variable population size (13 mins) as with fixed population size (45 mins). If we have opted for a fixed population size of 100 individuals, here we would opt for the initial population size of 50 individuals. First, shorter makespan (3026 vs. 3030 mins) has been achieved by a smaller population size and second, and more important, a shorter makespan has been achieved by a considerably shorter runtime (228 vs. 358s).

3.5 Crossover rate

The final experiment was performed to determine the best CUT and LOX crossover rate with FRD selection, EX+SH+CUT+LOX methods, and a variable population size with initial value of 50 individuals (Fig. 16). Ten optimization runs were performed for each BOM and for each combination of methods. For each run, the optimization was performed until there was no makespan improvement for ten consecutive generations. We have used EX variable mutation rate in this experiment. Again, the makespan is represented by a solid line and runtime by a dashed one.

The shortest makespan (3026 mins) was achieved if 60% CUT crossover rate was applied; the runtime was 221s. The longest timespan was achieved by a CUT crossover rate of 0%, i.e. when only LOX crossover was applied (3069 mins); in this case, runtime was also the longest – 255s. The timespan between the shortest and the longest makespan is 43 mins. The CUT crossover rate with the shortest runtime (207s) is 100%, yet the makespan (3052 mins) is significantly slower than the best one (3026 mins). If we opt for a 14s slower runtime (60% CUT crossover rate) we benefit 26 mins regarding makespan.

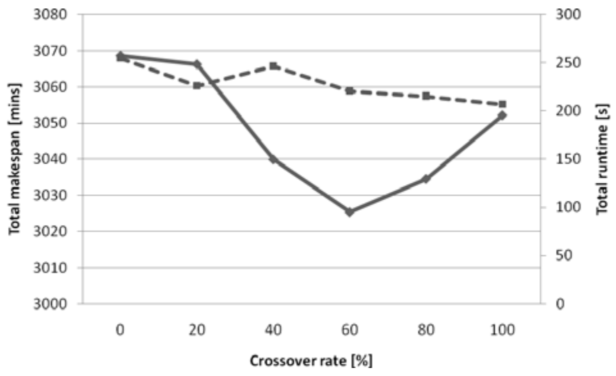


Fig. 16. The comparison of results of the GA with FRD selection, EX+SH+CUT+LOX methods, and a variable population size, regarding the CUT crossover rate.

4 Conclusion

Results achieved in this preliminary study are promising since only the application of different GA operators had reduced the initial total makespan from 3489 mins to 3026 mins, thus reducing the makespan by 15%. We could opt also for an even higher reduction to 3013 mins but on the account of significantly higher optimization runtime (661s instead of 221s). However, our opinion is that improvement of makespan by 0,7% could not be justified by increasing the optimization runtime by almost 300%. Nevertheless, it is the end-user that makes the final decision, but since daily optimization is required, the end-user would probably also opt for a shorter runtime and the makespan a bit longer. To emphasize the importance of a tradeoff between makespan and runtime, it must be stated that the optimization was done with only three BOMs. If more BOMs would be introduced, the optimization runtime would increase greatly, since the JSS problem would become even more complex and time consuming, yet very challenging to provide a solution implemented in an actual production process.

However, still there are many GA operators that need to be explored. Also, the initial population could be generated by another method to provide an initial population that would produce even better results. Furthermore, GAs are only one method to solve JSS problem. There are many more methods that successfully solve such kind of problems, e.g. tabu search and ant colony optimization.

The opposite-based concept is also intriguing and worth examining into detail in the scope of JSS. Our research has proved that mirroring yields a fitter initial population. Whether mirroring is the same as the opposite-based concept remains unanswered, because it is often hard to find an opposite, e.g. the opposite of a chess move. The same goes for the job shop schedule. Therefore, this intriguing problem is left for future research.

***Acknowledgement**: This research was supported by the Ministry of Higher Education, Science and Technology of the Republic of Slovenia (Contract No. Z5-0015-0586-08). Our sincere thanks goes also to Mr. Bolčič and his team who have initiated this research.*

References:

[1] Bernik I., Multicriteria scheduling by applying genetic algorithms and Petri nets visual simulation, *Doctoral Dissertation* (in Slovene), University of Maribor, Faculty of Organizational Sciences, 2001.

[2] Chen H., Ihlow J., and Lehmann C., A Genetic Algorithm for Flexible Job-Shop Scheduling, *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*, Detroit, Michigan, May 1999, pp. 1120-1125.

[3] Chiu Hung-Pin H.P., Hsieh K.L., Tang Yi-Tsung Y.T., and Chien Wan-Jung W.J., Employing a genetic algorithm based on knowledge to address the job shop scheduling problem, *WSEAS*

Transactions on Computers Research, Vol. 2, No. 2, 2007, pp. 215-221.

[4] Choi S.H. and Yang F.Y., A filter search algorithm based on machine-order space for job-shop scheduling problems, *Journal of Manufacturing Technology Management*, Vol. 17, No. 3, 2006, pp. 376-392.

[5] Eiben A.E., Hinterding R., and Michalewicz Z., Parameter control in evolutionary algorithms, *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 2, 1999, pp. 124-141.

[6] Eiben A.E. and Smith, J.E., *Introduction to Evolutionary Computing*, Springer, 2003.

[7] Eiben, A.E., Marchiori, E., and Valko, V.A., Evolutionary algorithms with on-the-fly population size adjustment, *Lecture Notes in Computer Science*, Vol. 3242, Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference Proceedings, 2004, pp. 41-50.

[8] Fowler A., Systems modeling, simulation, and the dynamics of strategy, *Journal of Business Research*, Vol. 56, No. 2, 2003, pp. 135-144.

[9] Gen M. and Cheng R., *Genetic Algorithms & Engineering Design*, John Wiley & Sons, Inc., 1997.

[10] Ho N.B. and Tay J.C., GENACE: an efficient cultural algorithm for solving the flexible job-shop problem, *Proceedings of the 2004 Congress on Evolutionary Computation*, Vol. 2, 2004, pp. 1759-66.

[11] Hyung R.C. and Byung J.P., Genetic algorithm for the integration of process planning and scheduling in a job shop, *WSEAS Transactions on Information Science and Applications*, Vol. 3, No. 12, 2006, pp. 2498-2504.

[12] Kljajić M., Bernik I., and Škraba A., Simulation approach to decision assessment in enterprises, *Simulation*, Vol. 74, No. 4, 2000, pp. 199-210.

[13] Naso D., Surico M., Turchiano B., and Kaymak U., Genetic algorithms for supply-chain scheduling: A case study in the distribution of ready-mixed concrete, *European Journal of Operational Research*, Vol. 177, No. 3, 2007, pp. 2069-2099.

[14] Oliveira J.A., Scheduling the truckload operations in automatic warehouses, *European Journal of Operational Research*, Vol. 179, No. 3, 2007, pp. 723-735.

[15] Ong Z.X., Tay J.C., and Kwoh C.K., Applying the Clonal Selection principle to find Flexible Job-Shop schedules, *Lecture Notes in Computer Science*, Vol. 3627, Artificial Immune Systems: 4th International Conference, ICARIS 2005, pp. 442-455.

[16] Rahnamayan S., Tizhoosh H.R., and Salama M.M.A., Opposition-Based Differential Evolution, *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 1, 2008, pp. 64-79.

[17] Reeves C., A genetic algorithm for flow shop sequencing, *Computers and Operations Research*, Vol. 22, 1995, pp. 5-13.

[18] Shi, X.H., Wan, L.M., Lee, H.P., Yang, X.W., Wang, L.M., and Liang, Y.C., An improved genetic algorithm with variable population-size and a PSO-GA based hybrid evolutionary algorithm, *International Conference on Machine Learning and Cybernetics*, Vol. 3, 2003, pp 1735-1740.

[19] Wang Q. and Chatwin C.R., Key issues and developments in modelling and simulation-based methodologies for manufacturing systems analysis, design and performance evaluation, *The International Journal of Advanced Manufacturing Technology*, Vol. 22, No. 8, 2004, pp. 720-729.

[20] Yoshikawa M. and Terai, H., Genetic algorithm engine for scheduling problems, *WSEAS Transactions on Circuits and Systems*, Vol. 5, No. 3, 2006, pp. 397-402.