ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20161207195125
Original scientific paper

Production Planning and Optimization of Work Launch Orders Using Genetic Algorithm

Tomislav BRAJKOVIĆ, Mladen PERINIĆ, Milan IKONIĆ

Abstract: This paper presents genetic algorithm approach for simultaneous production planning and optimization of work launch orders. Genetic algorithm was used to optimize overall quantities of each machine, work launch orders and the number and size of lots, which are going to be launched into production in each operative period. Simulation of work launch orders and quantities of each machine was performed using linked lists of empty spaces. Optimal machine quantities were achieved using linked lists of empty spaces combined with the gene editing procedure. This study has shown that the proposed genetic algorithm can find a shorter production cycle of production assortment inside the operative period, using smaller machine quantities.

Keywords: genetic algorithm; optimization; production cycle; production planning; work launch orders

1 INTRODUCTION

In general, scheduling is the allocation of shared resources over time to competing activities in such a way that predefined performance measure is optimized [1], [2]. Scheduling problems arise in different areas such as in the assembly of electronic components on printed circuit boards (PCB's), vehicle routing (VRP), scheduling of data channels in communications, various resource scheduling, flexible manufacturing systems and so on. All of these problems fall in the category of nondeterministic polynomial (NP) hard problems. This means that if someone can offer solution for granted, such solution can be tested and verified in polynomial time, but there is no efficient solution algorithm which can solve it to optimality in polynomial time [3]. Description of NP-completeness can be found in [4], and study on NP-hardness of shop scheduling problems can be found in [5].

Talking about machine scheduling problems usually means talking about Multi-Stage Multi-Machine problems: 1). Flow-shop, 2). Open-shop, 3). Job-shop and 4). Group-shop Environment. Job-shop scheduling problem (JSSP) can be described informally as follows [3]: There is a set of (*n*) jobs (products) and set of (*m*) machines. Each job consists of a chain of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine, and each machine can process at most one operation at a time. Usually the objective is to minimize the make-span, i.e. the time required to complete all jobs.

Classical job-shop model puts several constraints on jobs and machines: 1) A job does not visit the same machine twice, 2) There are no precedence constraints among operations of different jobs, 3) Operation cannot be interrupted, 4) Each machine can process only one job at a time and 5) Neither release times nor due dates are specified. In the classical JSSP two kinds of constraints are placed on the operations: a) operation precedence constraint for a given job and b) operation un-overlapping constraint for a given machine [6].

JSSP is not only NP-hard, but it is one of the hardest combinatorial problems. Hence, the scientists have developed approximation methods and heuristics that give solutions not far from optimum, but in reasonable time.

Implementation of branch and bound procedure for solving JSSP can be found in [7]. Authors used special heuristic methods to generate a sequence (search tree) of initial (complete) solutions, where each new solution (node) is obtained from the immediately preceding solution by moving one operation of some segment before its first or after its last operation. The authors have presented computational results for 100 operations (10 jobs on 10 machines and 20 jobs on 5 machines). Study has shown that such procedure can be used for many scheduling problems. Another approach for dealing with JSSP can be found in [8]. Authors have tackled the JSSP by replacing discrete jobs with the flow of continuous fluid, where the feasible schedule is constructed from the fluid relaxation. Study has shown that proposed algorithms make no probabilistic assumptions, and are asymptotically optimal for all instances with a large number of jobs. Results of the paper imply that the combinatorial structure of the problem becomes increasingly unimportant because the problem is well approximated by continuous fluid relaxations which are efficiently solvable. Use of Lagrangian relaxations (LR) to schedule shops can be found in [9] and [10]. LR are used to decompose the original NP-hard problem into smaller sub-problems which are easier to solve. Near-optimal schedules are obtained by iteratively solving those sub-problems and updating Lagrangian multipliers at a higher level. Both papers have shown that presented LR methods can generate high quality schedules in a satisfactory time, and because of decomposition, individual parts can be solved on distributed system. This means that even large-scale JSSP could be solved effectively even in a rapidly changing environment. Additional example of using various mathematical models for flexible job-shop scheduling problem (FJSSP) can be found in [2]. In [11] authors developed a mixed algorithm based on a non-delay scheduling heuristic and working time window algorithm. Study has shown that the result of such algorithm is better than the result obtained by the algorithm that ignores machine availability constraints. Example of combining methods to solve JSSP can be found in [12]. Authors combined Monte Carlo simulation, solutions to the associated problem, with either constraint programming or Tabu search. Research has shown that such combination results in algorithm that scales best both in terms of problem size and uncertainty. Speed comparison of proposed algorithm versus established heuristics for the flow-shop scheduling problem can be found in [13]. Developed algorithm can go side by side with the existing methods, but as the authors have stated, it was far simpler to implement. That paper can serve as the basis for further study of JSSP and associated algorithms.

Genetic algorithm (GA) is a stochastic search technique that uses mechanism of natural selection and natural genetics [1, 3, 14, 15, 16]. In general, population consists of chromosomes, where each chromosome represents solution to the problem. Chromosomes evolve through successive iterations called generations. Each chromosome in each generation is evaluated using some measure of fitness. Best chromosome (solution) is usually kept until dethroned by a better solution (elitism). Next generation is formed usually through some kind of cross over and mutation process. After several generations, the algorithm should converge to the optimum or sub-optimal solution relatively close to optimum.

In [17] authors used critical block neighborhood (CBN) transition operator. CBN permutes the order of operations in critical block by moving the operation to the beginning of the critical block. Research has shown that the combination of critical block, disjunctive graph distance and GA can provide result on par with the compared methods. Another example of hybridization can be found in [18], [19] and [20]. In [18] authors combined GA, schedule generator procedure and a local search procedure. First, GA was used to define and evolve the priorities of the operations and delay times, then in the construction procedure priorities and delay times were utilized for construction of parameterized active schedules, and in the end, local search procedure (disjunctive graph and the neighborhood) was used to improve the solution obtained by the construction phase. Proposed algorithm produced optimal or near-optimal solutions. Overall, the algorithm produced solutions with average relative deviation of 0.39% to the best known solution. Job-shop scheduling problem with sequence-dependent setup times was solved in [19] using GA, disjunctive graph representation and local search (neighborhood and hill-climbing based on make-span estimation) method. Proposed approach has outperformed all the other methods used in that paper. Another example that shows the benefits of hybridization can be found in [20]. Hybrid approach using GA, Tabu search and simulated annealing in [20] has produced better results than all of the tested algorithms. Hybrid algorithm was capable of yielding optimal solutions with the average make-span values lower than those obtained by the other algorithms. In all of the tests, proposed hybrid has also reached optimal values in smaller number of iterations.

Power of the GA can be seen in [21]. Authors used pure GA trying to keep genetic operations as simple as possible. The algorithm did not manage to find the optimal solution for the considered problem, but even a very simple GA like the one presented in that paper was capable of finding good solutions. Authors stated that because genetic algorithms are not well-suited for fine-tuning of solutions around optima, the next step should be some form of hybridization. Authors in [22] did not use hybridization, but they did present GA using "fuzzy roulette wheel selection", hierarchical clustering crossover and a new mutation operator. Proposed algorithm had the best relative error among the compared methods. Research has shown that proposed selection, crossover and mutation operations significantly improved the performance of proposed model when used in place of the original operations typically used in GA. Another example of using GA for solving JSSP can be found in [23].

Purpose of this research is to develop and present complete solution for simultaneous production planning and optimization of work launch orders, based on GA in combination with linked lists of empty spaces (idle times).

2 DESCRIPTION OF THE PROBLEM AND CREATION OF CHROMOSOMES

GA approach in this paper was based on the requirements and assumptions as follows. *Operative period* is one work week (5 days times 2 eight hours shifts equals 80 hours). *Operative quantity* is the total number of units of individual product (job) that needs to be produced inside the operative period. *Lot* (tactical quantity) ranges from 1 to operative quantity, and shows in how many lots (batches) is each operative quantity going to be launched into production. Goal is to optimize work launch orders and to find minimal number of machines able to produce the operative quantities by the end of each operative period.

Table 1 General information used for production planning

Machine no.		Name of the machine	Machine label
1		lathe	0
2		induction hardening device	1
3		grinder	2
4		mill	3
5		drill	4
Product (job)		Order of operations for each product	Total number of operations
Name	Label		
A	0	0 - 1 - 2	3
B	1	0 - 3 - 1 - 2	4
C	2	0 - 4 - 3 - 4	4
		Required number of units at the end of each operative period (80 hours)	
A	0	579	
B	1	965	
C	2	385	

On such production system three products are going to be made: product A, B and C. Tab. 1 shows general information used in production planning. Duration of operations per one unit is given in Tab. 4 in the Appendix A. Each chromosome consists out of three parts as shown in Fig. 1. First part represents number of lots of each product, second part represents number of each machine and third part represents order of operations named through the number of each lot. How to distinguish among the operations with the same label is explained in chapter 3. Description of the algorithm.

2.1 Creation of the 1st Part of the Chromosome

First part of the chromosome (DNA, Fig. 1) is created by randomly choosing the number of lots in which each product is going to be launched. Tab. 2 shows all possible combinations of number of lots and units per lot.

Table 2 Number of lots and units in each lot

Product	Operative quantity	Tactical quantity (number of lots)	Lot size
A	579	1	579
		3	193
		193	3
		597	1
B	965	1	965
		5	193
		193	5
		965	1
C	385	1	385
		5	77
		7	55
		11	35
		35	11
		55	7
		77	5
		385	1

Imposed restriction in this paper was that each lot of a certain product must contain same number of units (no uneven lots). In Fig. 1 the area colored black (1st part of the chromosome) represents number of lots for each product. Product A (at index 0 of the 1st part) consists out of 3 lots (193 units per lot), product B (at index 1 of the 1st part) out of 5 lots (193 units per lot), and product C (at index 2 of the 1st part) out of 5 lots (77 units per lot) as shown in Tab. 2. This process of randomly choosing the number of lots is carried out for each chromosome and determines the number of genes, i.e. the length of the 3rd part of the chromosome, which will be explained further along. Sum of the number of lots for all products in the example (3+5+5) gives total number of lots (13).

2.2 Creation of the 2nd Part of the Chromosome

Second part of the chromosome is created by randomly choosing total number of each machine (from 1 as the lower bound to some predetermined number as the upper bound). When choosing the upper bound of machines, the only restriction is that it should be "sufficient enough". Choosing the "sufficiently enough" upper bound can be problematic, because the selected value could be less than the optimal number of each machine (search space could be too small). Proposed solution for this problem is to use the method philosophically similar to gene editing technology, as explained in the later chapter.

2.3 Creation of the 3rd Part of the Chromosome

Length of the 3rd part of the chromosome (number of genes) depends on the number of lots randomly chosen in the 1st part of the chromosome (Fig. 1). It is calculated according to Eq. (1).

$$Nog = \sum_{i=0}^{n} b_i \cdot no_i \tag{1}$$

In Eq. (1) *Nog* denotes number of genes, *b* is the number of lots of product *i* (Tab. 2) and *no* is the total number of operations in product *i* (Tab. 1). Using Eq. (1) the number of genes for the example in Fig. 1 is calculated as (Eq. 2):

$$Nog = 3 \cdot 3 + 5 \cdot 4 + 5 \cdot 4 = 49 \tag{2}$$

After calculating the number of genes, the name of the operation (machine) in each product in each lot is swapped with the number (name) of the lot, i.e. lot-based representation. For example, product B (1) has four operations (0-3-1-2 as shown in Tab. 1), and is being launched into production in five lots (1st part of chromosome at index 1 in Fig.1). Every lot of product B is given a number from [0-12] (total of 13 lots). Five lots of product B can be represented for example with numbers 1, 4, 7, 9 and 11. The only restriction on the numbering of lots is that one lot corresponds only to one of the thirteen numbers [0-12]. Every operation in product B is swapped with the lot number. Operations 0-3-1-2 become 1-1-1-1, 4-4-4-4, 7-7-7-7, 9-9-9-9 and 11-11-11-11 (genes colored blue in Fig. 1). After repeating this procedure for all the products, genes (operations of products represented through the name of the lot) are shuffled and placed in the 3rd part of the chromosome (Fig. 1).

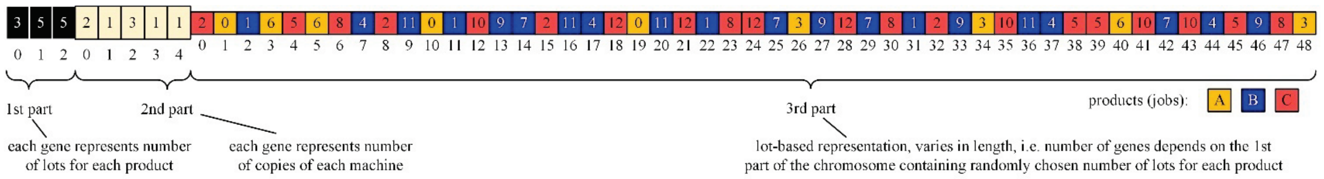

Figure 1 Representation of the chromosome (DNA, genotype)

3 DESCRIPTION OF THE ALGORITHM

Entire program was implemented using C++ language. Fig. 2 represents flow chart of the solution algorithm.

Initialization and creation of multiple populations (islands) are carried out as the first steps. As stated, the 1st part of the chromosome determines the length (number of genes) of the 3rd part. This means that the populations with different numbers (lots) in the 1st part will have different memory requirements (each gene is represented with 32 bit integer). If the chromosomes with different lots were in the same population, whole population could not be represented as an array (1D in this research) of fixed length, because the size of that array would very likely change with every new generation. Furthermore, there is no point in crossover of chromosomes with different lengths, because such crossover would result in chromosomes (offspring) with the new number of lots. New number of lots does not guarantee that the optimal order of the operations (genes) in the 3rd part of parent chromosomes would be the same for the offspring, when the number of lots changes after the crossover of the 1st part (Fig. 3).

If the two chromosomes of different lengths in Fig. 3 produce offspring using for example order-based crossover, such crossover procedure would resemble more to "shuffling" of genes than steady evolution, i.e. gradual progression towards optimal solution. Four out of five lots of product B (those at index 1 of the 1st part of the parent chromosome 1) would not exist in child chromosome. However, the optimal order of genes in parent one (starting with lot 2 at index 0 in the 3rd part) was made with all five lots of product B in mind, and not just with the one lot that "fits" in the child chromosome. Additionally, every population can be run in its own thread, so it is more friendly from the separation of concerns point of view.

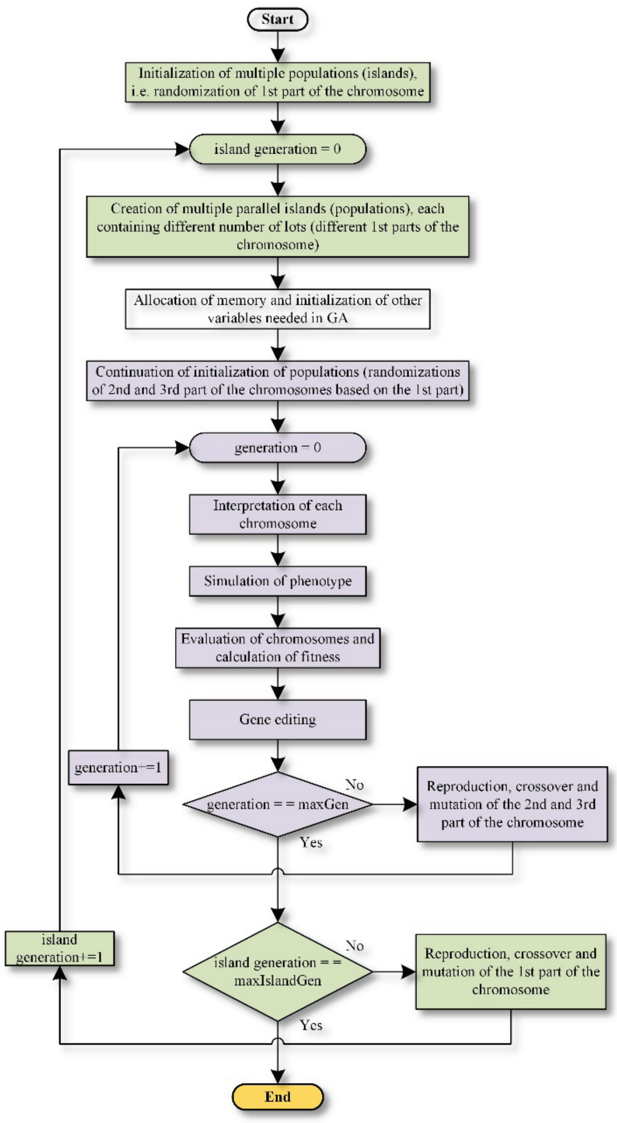
Start
Initialization of multiple populations (islands), i.e. randomization of 1st part of the chromosome
island generation = 0
Creation of multiple parallel islands (populations), each containing different number of lots (different 1st parts of the chromosome)
Allocation of memory and initialization of other variables needed in GA
Continuation of initialization of populations (randomizations of 2nd and 3rd part of the chromosomes based on the 1st part)
generation = 0
Interpretation of each chromosome
Simulation of phenotype
Evaluation of chromosomes and calculation of fitness
Gene editing
generation+=1
generation = = maxGen
No
Yes
Reproduction, crossover and mutation of the 2nd and 3rd part of the chromosome
island generation+=1
island generation = = maxIslandGen
No
Yes
Reproduction, crossover and mutation of the 1st part of the chromosome
End

Figure 2 Flow chart of the proposed algorithm

Computation of all the necessary variables and allocation of memory can be carried out only after the randomization of lots. After that, 2nd and 3rd part of the chromosomes can be randomized in each population (island).

Interpretation of chromosomes is a procedure in which every gene of the 3rd part is associated with a unique code (a unique number). First, 1D array is created for every chromosome in the population. That array has the same length as the 3rd part of the chromosome, and can be identified with the mRNA. Then, a unique number for each gene is calculated. This number is stored in that 1D array at the same position (index) as the gene which it was calculated for. This unique number serves multiple purposes.

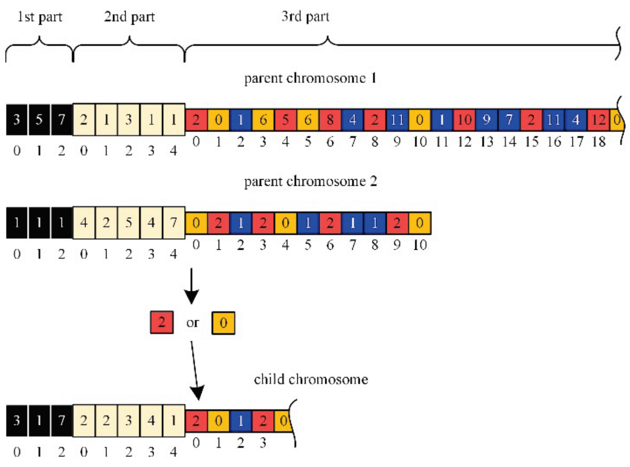

Figure 3 Shuffling of genes resembling crossover

First, it codes [operation-lot-machine visit] triplet. This was necessary to uniquely distinguish between different operations on the same machine, and to bind genes of the 3rd part with the machine (operation). For example, product C (2) has the order of operations 0-4-3-4 (Tab. 1). When the name (number) of operation is substituted with the name (number) of lot, one possible outcome could be 5-5-5-5. Not only that it is required to distinguish amongst the fives which operation they should represent, but it is also required to distinguish between the two fours (0-4-3-4) using the notation with fives. Although number four refers to the same operation (to the same machine), first operation four and second operation four can last differently. First, the requirement is to drill five holes, then it is needed to mill some part, and after the milling requirement is to drill additional two holes. Both operations are drilling, they are done on the same machine (same operation, same number four), but they last differently. Here unique codes come in to help distinguish between these and other similar conundrums. Another purpose of these codes is that they represent addresses (indexes) in the data structure which will carry all the necessary information. Information like the beginning and the end of the operation, duration, affiliation to lot, product and the machine number on which operation goes, set-up times etc.

After the mRNA has been constructed for every chromosome, construction and simulation of phenotype is carried out. In this paper, this was done using linked lists of free (empty) spaces of each machine (Fig. 4).

In the beginning, each machine (each copy) gets its own pointer to a linked list, and the first and only element in that list is an empty space. This empty space represents time line from zero to "infinity". Somewhere in that timeline operations will be launched and will end. Starting from the beginning, genes are read one by one from the 3rd part of the chromosome, they are interpreted with the help of mRNA and are put in the correct location in the linked list structure. Search for the empty place to put the operation (gene) into, goes from the first copy (numbered with 0) to the last copy of the affiliated machine. Also, search for the empty place always starts at the beginning of the linked list of each copy and lasts until the first available empty space is found. Of course, that "first" empty space must fulfill all the restrictions like operation precedence and interrupt constraints, one machine performs only one job at a time constraint, parallel or gradual cycle of production constraint, operation un-overlapping for a given machine constraint and so on.

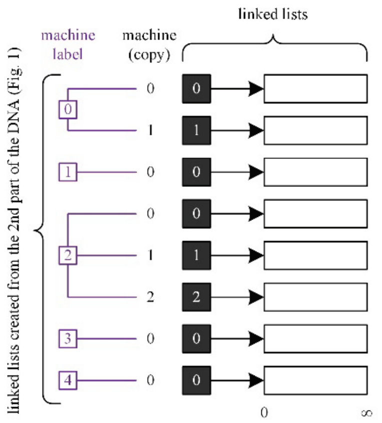

Figure 4 Creation of empty linked lists for each copy of every machine

For example, Fig. 5 a) shows snapshot somewhere in the process of simulation of the 3rd part of the chromosome. Digression, operations 01-02 (grey) are added in for the purpose of example and are not the part of the example chromosome from Fig. 1. The operation of interest is operation 32 colored blue in Fig. 5 a). All other operations (01-02, 11-12, 21-22 and 31) are already in place dictated by the order of genes (operations expressed through the name of the lot). The next operation (gene) that needs to be placed is operation 32. Because operation 32 needs to go onto machine 2, and machine 2 has three copies (0, 1, 2), operation 32 is simulated on all of the three copies as shown in Fig. 5 b).

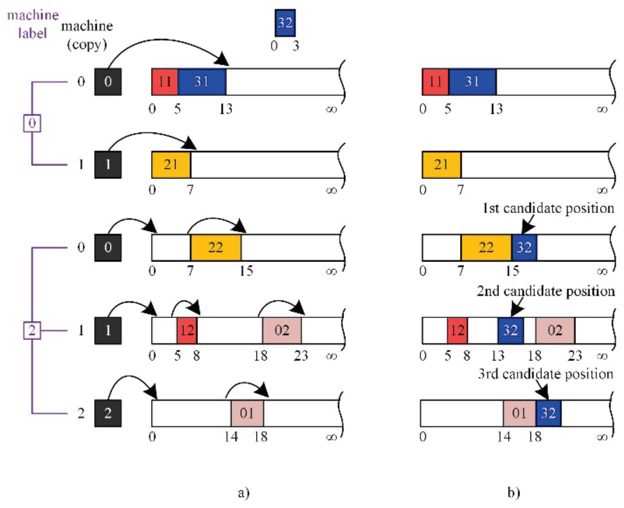

Figure 5 Snapshot of the simulation process

Out of those three candidate positions (copies of the machine), 2nd candidate is chosen as the best one, because that candidate position on 1st copy of the machine 2, assures the earliest possible start of operation 32 (start time of 13) in the context of the example. Same principle of choosing the start time applies for all the operations in real simulation. During the process of simulation all permissible left shifts are explored, and are guaranteed by the nature of linked list search algorithm of empty spaces (complying with all the restrictions placed on the operation and the machine). The end result is active schedule for that chromosome. If multiple copies of the machine can start and finish some operation at the same earliest time, during the simulation process that operation is distributed on the first available copy of the machine (first linked list) that can start and finish that operation at the earliest possible time. After all the genes (operations) are placed into their earliest starting positions obeying all restrictions, earlier mentioned data structure now holds all the information needed to construct the resulting organism, i.e. phenotype (Gantt chart).

In this paper, evaluation of chromosomes (fitness function) is focused on objectives to minimize the number of machines (copies of each), production cycle, number of lots, and to satisfy the eighty hour deadline.

After the chromosomes have been evaluated, fitness has been assigned, and before the procedures of crossover and mutation, a gene editing procedure is being performed on the 2nd part of the chromosome. The example is as follows. When creating the population, values of genes (number of copies of each machine) in the 2nd part of the chromosome (Fig. 1) are being randomized, between values of 1, and "sufficiently enough" large number. To be on the safe side, this "sufficiently enough" number could be for example ten thousand (10 000). Ten thousand of each copy of the machine should be enough to handle the eighty hour deadline. If it is not, this number can be raised to any value (obeying computer memory limitations).

After the very first generation, gene editing will correct upper bound. Left side of Fig. 6 shows two parents with randomized 2nd part.

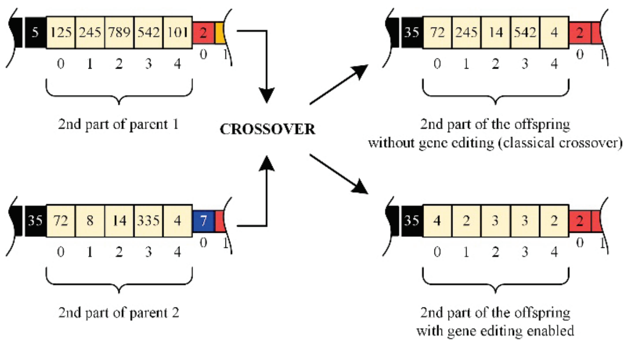
2nd part of parent 1
CROSSOVER
2nd part of the offspring without gene editing (classical crossover)
2nd part of parent 2
2nd part of the offspring with gene editing enabled

Figure 6 Crossover with and without gene editing of the 2nd part the chromosome

As stated, number of copies in each parent goes from 1 to 10 000. Using the classical approach, crossover could result in a child as shown in the upper right corner of Fig. 6. But during the simulation of the 3rd part of the chromosome using linked lists, the simulation has shown that for example, out of 789 available copies of machine 2 (grinders at index 2 of the 2nd part of parent 1), only 3 where filled with operations, and the other 786 copies (other 786 linked lists) were empty, i.e. containing only one (original) empty space. Simulation has also shown that out of 14 available copies of grinder in parent 2 (at index 2), only 4 of them are assigned with operations, other 10 machines (linked lists) were empty. This is where gene editing comes in. Instead of using classical crossover which could result in an offspring as in the upper right part of Fig. 6, only the machines that have operations on them (the ones whose linked lists are not empty) are used to calculate the real number of copies. The rest of the copies are discarded. Because this is the very first generation (first crossover), selected number of copies is the worst case scenario, and the number of copies needed to fulfill the 80 hour deadline should only go down as the evolution progresses. Crossover with edited genes can produce an offspring as the one in the lower right part of Fig. 6. This procedure of gene editing assures that the memory requirements are kept at the minimum, and that the genes of the 2nd part pass only useful information to the next generation. In this paper, mutation of the genes of the 2nd part is not suppressed, meaning that genes can mutate to all the values in the range (for example from 1 to 10 000), but the same simple procedure of gene editing, i.e. extracting useless information (useless copies) before each crossover, guaranties that only used copies of each machine participate in the evolution process.

4 TEST RESULTS

Original Gantt chart column in Tab. 3 shows the number of each machine, number of lots and production cycle end time overall (last lot end time) used in a real life production. This is the benchmark, and the goal of this research was to improve upon those results. Fig. 7 shows Gantt chart obtained with the proposed algorithm with the fixed number of lots and fixed number of machines. The only goal was to research if the proposed algorithm was able to find shorter production cycle. Algorithm was able to find schedule ending in 75.681 h, which is an improvement over the original Gant chart (Tab. 3). Then the number of copies and number of lots was not fixed. Gantt chart and number of copies obtained by the automatic selection are shown in Fig. 8 and in Tab. 3. Solution for the optimal number of lots obtained by the proposed algorithm was the same (3, 5 and 5), but the optimal number of the machines was reduced by one. Number of required copies of machine 3 (mill) was reduced from 2 to 1. This decrease resulted in the longer production cycle (from 75,681 to 77,037 h), but that time does not break the imposed restriction of 80 hours, and it is also shorter than the time obtained from the original Gantt chart.

Table 3 Results of the proposed algorithm

				Results obtained by the algorithm	
			Original Gantt Chart	Fixed number of copies and lots	Automatic selection of copies and lots
	Machine label				
Number of each machine	0		2	2	2
	1		1	1	1
	2		3	3	3
	3		2	2	1
	4		1	1	1
	Product (job)				
	Name	Label			
Number of lots	A	0	3	3	3
	B	1	5	5	5
	C	2	5	5	5
Production cycle end time overall (last lot end time) / h			79,93	75,681	77,037

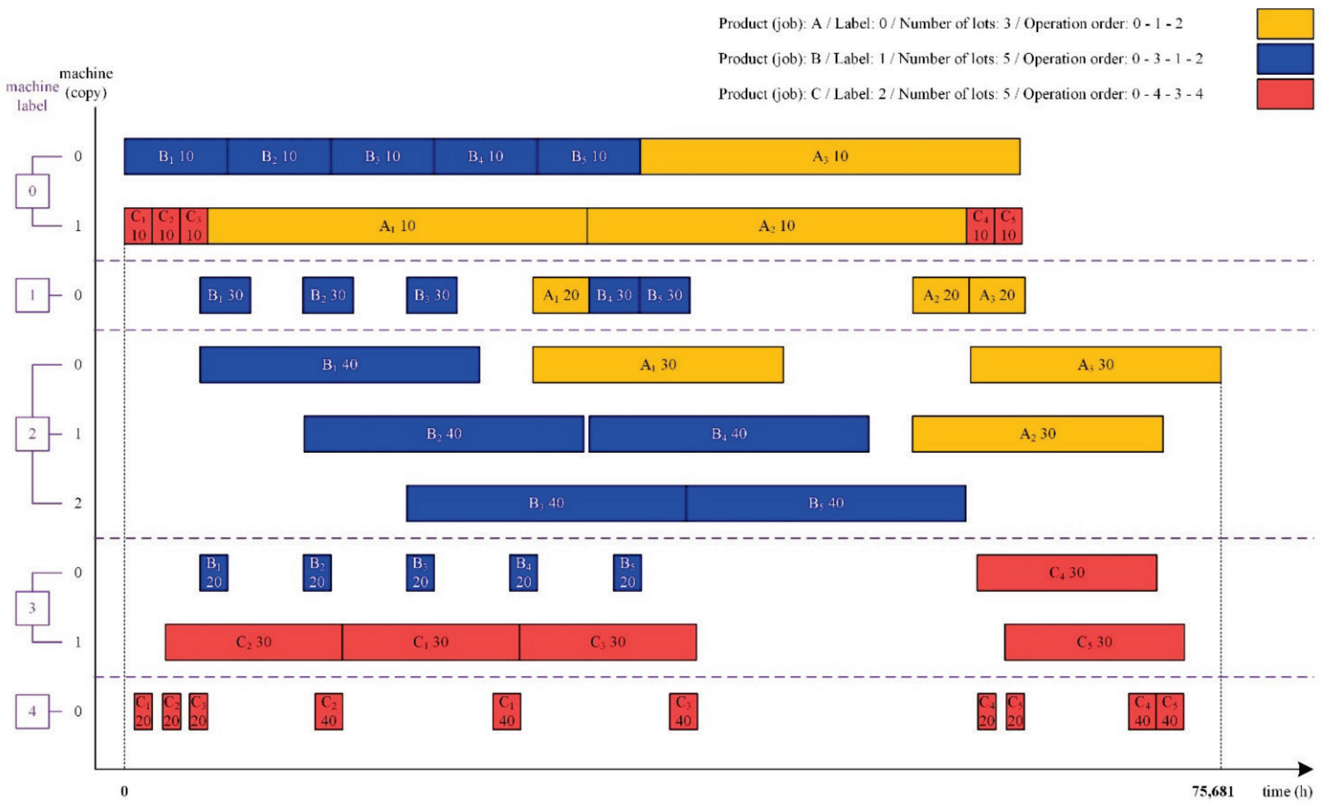

Figure 7 Gantt chart obtained with the proposed algorithm; number of lots and copies of each machine and was fixed

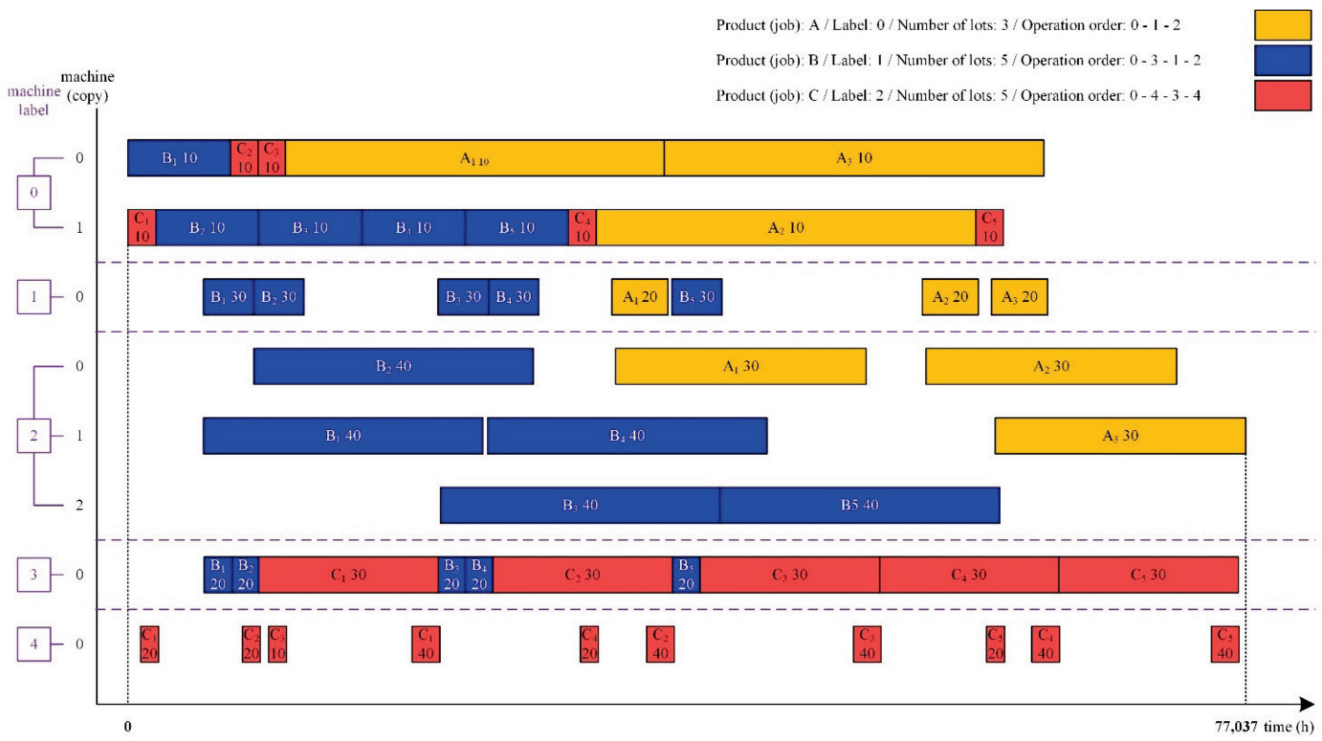

Figure 8 Gantt chart obtained with the proposed algorithm; algorithm was used to find the optimal number of lots and copies of each machine

5 CONCLUSION

This paper presents genetic algorithm for production planning and optimization of work launch orders using linked lists of empty spaces (linked lists of machine idle times). Linked lists represent natural way of looking on available time in which machines could execute some task. In reality, machine can only execute a certain task when it is idle (empty). So, any task can be executed only by using one of the available idle times (elements of linked list). Not only that each element of linked list signalizes machine idle time (emptiness), but it also shows the duration of that idle time, and more important its position in operative period. Properties (start, finish, duration, position and inclusion of position in simulation process) of each element in the linked list can be easily manipulated. Such manipulation of linked list elements (idle times) in operative period can be used in the selection process of different tactics for launching work orders, and various other decision making processes. Each element of the list can also contain various useful info. This helps that all information about respective idle time stays compact. Number of elements in linked list shows how chopped up is the total time availability (capacity) of machines. Large number of elements indicates toward which machines optimization efforts should be directed. With the number of copies and number of lots fixed, proposed algorithm was capable to find much shorter production cycle. With all the values unlocked, algorithm was able to decrease total quantity of required machines by 1 machine. Because the deadline was not broken and the number of machines was reduced, solution obtained with proposed algorithm represents better choice in the context of this research. Gene editing before each crossover had the greatest impact on the reduction of machine quantity. This is because of its constant monitoring and deletion of unused machines, which assures that only the smallest number of copies of each machine participates in the crossover (evolution). This research has also shown that this kind of approach using genetic algorithm in conjunction with linked list of empty spaces can be used as a complete solution for production planning and optimization of work launch orders.

6 REFERENCES

[1] Yamada, T. & Nakano, R. (1997). *Job-shop scheduling - Genetic algorithms in engineering systems*. London: The Institution of Electrical Engineers, 134-160.

[2] Özgüven, C., Özbakir, L., & Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling, 34*(6), 1539-1548. https://doi.org/10.1016/j.apm.2009.09.002

[3] Gen, M. & Cheng, R. (1997). *Genetic Algorithms and Engineering Design*. Ashikaga Institute of Technology, Ashikaga, Japan.

[4] Cormen, T. H. et al. (2009). *NP-Completeness. Introduction to Algorithms*. Third Edition, Cambridge, Massachusetts: The MIT Press, 1048-1105.

[5] Sotskov, Yu. N. & Shakhlevich, N. V. (1995). NP-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics, 59*(3), 237-266. https://doi.org/10.1016/0166-218X(95)80004-N

[6] Perinic, M. (2004). Optimization of production cycle of flexible manufacturing system using genetic algorithm. *Doctoral thesis*, Faculty of Engineering, University of Rijeka,

[7] Janiak, A. & Szkodny. T. (1994). Job-shop scheduling with convex models of operations. *Mathematical and Computer Modelling, 20*(2), 59-68. https://doi.org/10.1016/0895-7177(94)90207-0

[8] Bertsimas, D. & Gamarnik, D. (1999). Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms, 33*(2), 296-318. https://doi.org/10.1006/jagm.1999.1047

[9] Hoitomt, D. J., Luh, P. B., & Pattipati, K. R. (1993). A practical approach to job-shop scheduling problems.

Robotics and Automation, IEEE Transactions on Robotics and Automation, 9(1), 1-13. https://doi.org/10.1109/70.210791

[10] Wang, J., Luh, P., Zhao, X., & Wang, J. (1997). An optimization-based algorithm for job shop scheduling. Sadhana, 22(2), 41-256. https://doi.org/10.1007/BF02744491

[11] Ploydanai, K. & Mungwattana, A. (2010). Algorithm for Solving Job Shop Scheduling Problem Based on machine availability constraint. *International Journal on Computer Science and Engineering, 2*(5), 1919-1925.

[12] Beck, J. C. & Wilson, N. (2007). Proactive Algorithms for Job Shop Scheduling with Probabilistic Durations. *Journal of Artificial Intelligence Research*, 28, 183-232. https://doi.org/10.1613/jair.2080

[13] Modrák, V. & Pandian, R. S. (2010). Flow shop scheduling algorithm to minimize completion time for n-jobs m-machines problem. *Technical Gazette, 17*(3), 273-278.

[14] Sivanandam, S. N. & Deepa, S. N. (2008). *Introduction to Genetic Algorithms*, Springer-Verlag, Berlin, Heidelberg.

[15] Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, England.

[16] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization & Learning*, Addison-Wesley Publishing Company, INC., United States of America.

[17] Omar, M., Baharum, A., & Hasan, Y. A. (2006). A job-shop scheduling problem (JSSP) using genetic algorithm (GA). *Proceedings of the 2nd IMT-GT Regional Conference on Mathematics, Statistics and Application* / University Sains Malaysia, Penang.

[18] Gonçalves, J. F., Mendes, J. J., & Resende, M. G. C. (2002). A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem. *AT&T Labs Research Technical Report TD-5EAL6J*, http://www.optimization-online.org/DB_HTML/2002/09/538.html (23.1.2016)

[19] Gonzáles, M. A., Vela, C. R., & Varela, R. (2008). A New Hybrid Genetic Algorithm for the Job Shop Scheduling Problem with Setup Times. *Proceedings of the Eighteenth Int. Conference on Automated Planning and Scheduling* / Spain, 116-123.

[20] Thamilselvan, R. & Balasubramanie, P. (2012). Integrating Genetic Algorithm, Tabu Search and Simulated Annealing For Job Shop Scheduling Problem. International Journal of Computer Applications, 48(5), 42-54. https://doi.org/10.5120/7348-0283

[21] Lestan, Z., Brezocnik, M., Buchmeister, B., Brezovnik, S., & Balic, J. (2009). Solving job-shop scheduling problem with simple genetic algorithm. *International Journal of Simulation Modelling, 8*(4), 181-228. https://doi.org/10.2507/IJSIMM08(4)2.138

[22] Teekeng, W. & Thammano, A. (2012). Modified Genetic Algorithm for Flexible Job-Shop Scheduling Problems. *Procedia Computer Science, 12*(2), 122-128. https://doi.org/10.1016/j.procs.2012.09.041

[23] Perinic, M., Car, Z., & Mikac. T. (2004). Utilization of modified genetic algorithm (GJOB) for solving classical job shop scheduling problem. *Intelligent Manufacturing & Automation: Globalization – Technology – Men – Nature, Proceedings of the 15th DAAAM International Symposium* / Vienna, Austria.

Contact information:

Tomislav BRAJKOVIĆ
KEL d.o.o.
Stjepana Radića 86, 10312 Kloštar-Ivanić, Croatia
kel@zg.ht.hr

Mladen PERINIĆ
Faculty of Engineering,
Vukovarska 58, 51000 Rijeka, Croatia
mladen.perinic@riteh.hr

Milan IKONIĆ
Faculty of Engineering,
Vukovarska 58, 51000 Rijeka, Croatia
milan.ikonic@riteh.hr

Appendix A

Table 4 Duration of operations per one unit of product

	Operations (machine label)				
	0	1	2	3	4
Product	Duration of operations per one unit / h				
A (0)	0,136	0,02	0,09	-	-
B (1)	0,037	0,018	0,1	0,01	-
C (2)	0,025	-	-	0,16	1st. 0,017 2nd. 0,025