*Gazi University Journal of Science*
*GU J Sci*
*29(3):573-582 (2016)*

# Using Simulated Annealing for Flexible Robotic Cell Scheduling

G. Didem BATUR[1,♠], Serpil EROL[1]

[1]*Department of Industrial Engineering, Gazi University, Ankara, 06570, TURKEY*

*Received:11/05/2016 Revised:25/07/2016 Accepted: 05/08/2016*

**ABSTRACT**

In this study, the main focus is the scheduling problem arising in two-machine flexible robotic cells consisting of CNC machines in which sets of multiple part-types are produced. In such manufacturing cells, it is possible to process lots that contain different types of parts. The completion time of the production depends on the robot moves as well as the part assignments and processing times of the parts. The considered problem is to find the robot move sequence, the part sequence and the allocation of processing times of the parts on each machine, that jointly minimize the completion time of a particular set of production. A simulated annealing based algorithm is proposed in order to solve the problem of determining the best schedule in a two-machine cell. Experimental results show that this approach works well and can be extended to further cases.

***Keywords:*** *Flexible manufacturing systems; robotic cell; multiple part-type production; allocated processing times; simulated annealing.*

## 1. INTRODUCTION

As the competition in today's production systems increases, more flexible systems are needed in order to be successful. Automation level in such systems gets higher and, setup times are reduced to improve flexibility, resulting material handling time and cost being bottleneck. Due to these developments, the use of computer controlled machines and material handling devices become essential. Robots are commonly used in automotive, electric, electronic and machine industries. A robotic cell is a manufacturing cell consisting of a number of CNC machines and a material handling robot. Efficient use of these cells necessitates the tackling of some important and challenging problems.

Most of the studies about scheduling in robotic cells assume that each part being processed passes through the same sequence of locations [1, 2]. In such systems the only problem to be considered becomes to determine the part sequence together with the robot moves. In some other studies, researchers focused on the scheduling problem when all the machines are working in parallel. Under this headline, some researchers considered the identical machines case [3,4,5,6], whereas some others dealt with the non-identical parallel machine systems [7,8]. In our study, these two types of cells are combined under the heading of flexible manufacturing systems, adapting their properties into the definition of robotic cells. Cells that we considered consist of highly flexible CNC machines, which let the processing options to be adjusted. By the help of the definitions of operational and process flexibilities which state that one CNC can handle all of the operations of a part type and the processing sequence of these operations can be changed, it is possible to allocate every operation on any of the machines [9]. Such a cell involves the properties of both the flow shop and parallel machine environments, allowing the researcher to choose the most preferable option according to other parameters of interest. Gultekin

♠Corresponding author, e-mail: dbatur@gazi.edu.tr

et al. [10] dealed with the two machine, identical parts, operation allocation problem with tooling constraints. It is assumed that some operations can only be processed on the first machine while some others can only be processed on the second machine due to tooling constraints. Remaining operations can be processed on either machine. The problem was to find the allocation of the remaining operations to the machines and the corresponding robot move cycle that jointly minimize the cycle time. In our study, instead of assuming the process of a part to compose of a number of operations, we consider the total processing time to be composed of unit times and can be shared among machines. Jolai et al. [11] studied the problem of robotic cell scheduling with *m* machines with flexibility, load lock and swap assumptions. They considered the robotic cells producing parts of identical types repetitively and determined the cycle time of all 1-unit cycles in this type of robotic cell.

As we deal with robotic cells, we need to take robot movements, like the travel time from its existing point to the related machine and the loading/unloading times, into account. It is possible to consider setup operations that are assumed to be sequence-dependent in many studies, as a kind of such movements. Within this context, Mansouri et al. [12] addressed a two-machine flowshop scheduling problem to minimize setups and makespan where each job is characterized by a pair of attributes that entail setups on each machine. Naderi et al. [13] investigated scheduling job shop problems with sequence-dependent setup times under minimization of makespan and they developed a simulated annealing based metaheuristic to potentially solve the problem. Werner and Kravchenko [14] considered the problem of scheduling a set of jobs on a set of identical parallel machines, where a setup has to be performed by a given set of servers before the processing of a job can start. They assumed the processing of a job to be performed on one of the machines without interruption, and generalized some results for some specific cases. Mor and Mosheiov [15] studied on a single machine scheduling problem, where the machine is unavailable for processing for a pre-specified time period. They assumed that job processing times are position-dependent. Considering minimum makespan, minimum total completion time and minimum number of tardy jobs as the objective functions, they introduced simple heuristics which are based on solving the classical assignment problem. Jiang et al. [16] considered the scheduling problem on two parallel machines with a single server which is used for loading (setup) of the jobs before being processed on the machines. Allowing the processing time slot of each job to be preempted into a few pieces and these pieces to be assigned to possibly on distinct machines, this study is similar to our case in which we take the processing time allocation into account as one of the main problems. However, since we have two more decisions, namely the part assignments and the robot move sequences, our problem is more complex than this one.

We have considered the robotic cells which can process lots that contain different types of parts. Within the scope of multiple part type production, the decisions to be made include finding the robot move sequence and the part sequence, that jointly minimize the total production time. As we do not assume the allocation of processing times on each machine to be constant, we have a final decision which is the 'allocation'. As far as we know, this is one of the first studies to consider allocation possibility in multiple part-type robotic cell scheduling literature. In a previous study, Batur et al. [17] considered the problem of determining the best cyclic solution in a 2-machine manufacturing cells which repeatedly produce a set of multiple part-types, and where transportation of the parts between the machines is performed by a robot. They modeled the problem as a travelling salesman problem and then constructed a 2-stage heuristic algorithm, comparing the results with LPT.

Because of the hardness of these problems, mathematical models are usually inadequate throughout the solution procedures and it is very common for researchers studying on such subjects to prove the complexities and try to provide efficient algorithms. The rest of the paper is organized as follows. In the following section, we will define the problem and our notation. In Section 3, the metaheuristic approach will be defined and the algorithm developed will be explained. The computational results are reported in Section 4. Finally, Section 5 summarizes the contributions and concluding remarks of this study.

## 2. PROBLEM STATEMENT

In this study, flexible manufacturing cells consisting of CNC machines are considered. Robotic cells can process different types of parts which in general have different processing times. Considered problem is to find the robot move sequence and the part sequence minimizing the completion time of a particular set of production. During such a production, all the parts of an order enter the cell, get processed and leave the cell, returning the system to its initial state.

Based on the definitions of operational and process flexibilities; we consider an in-line robotic cell of two identical machines which are capable of performing all the required processes. Fig. 1 is given in order to represent such cells. There are two buffers in the structure; the one in the beginning is used as the input storage and the one in the end is used as the output storage. Each part is assumed to have known total processing times to be performed.

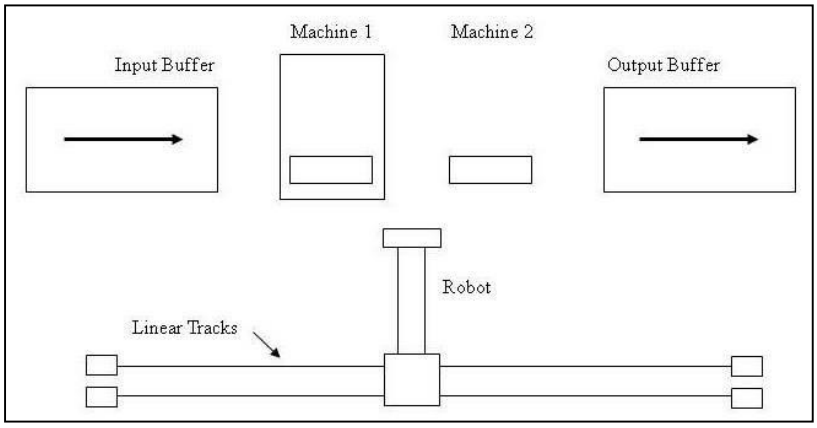



Fig. 1. In-line robotic cell layout

Taking the advantage of flexibility property, we claim that robot may choose either to perform all the processing of a part completely on any one of the machines or to share the total time among the machines. In order to use robotic cell systems efficiently, problems including the scheduling of the robot moves and the determination of the machines to perform processing of each part should be solved. We try to find the parts to be processed on the machines by allocating the processes to them and finding the robot move sequence which will jointly minimize the makespan. Throughout this study, we assume the processing times of parts to be integer valued.

Basic assumptions for this study that are common for most of the studies in the literature are as follows:

- All data are deterministic.

- Parts are always available at the input buffer and there is always an empty place at the output buffer.

- No buffer storage exists between the machines, each part is either on a machine or being handled by the robot.

- Neither the robot nor the machines can be in possession of more than one part at any time.

- The robot and the processing machines never experience breakdown and never require maintenance.

- Setup times are assumed to be negligible.

- No preemption is allowed in the processing of any operation.

As is mentioned above, our focus is on multiple part-type production. Thus, we need to solve both the problems of scheduling of parts and sequencing of robot moves for robotic cells. We do not make the assumption of the allocated processing time values on each machine being constant, thus allocation constitutes our third main problem.

## 3. SOLUTION PROCEDURE

### 3.1. Overview of SA

SA may be classified as an improvement procedure in nature. It was introduced by Metropolis et al. [18] and popularized by Kirkpatrick et al. [19] as a method to solve combinatorial optimization problems. Among the methods that have been developed to optimize various objectives like makespan, flow time, idle time, work-in-process and tardiness, etc., SA is believed to be the valuable search algorithm to accomplish these objectives [20].

Many researchers use simulated annealing to solve scheduling problems (e.g. see the work by Low et al. [21], Lee et al. [22], Behnamian et al. [23], Zhang and Wu [24]). Methods based on this approach have a remarkable ability that none of the traditional heuristic methods possess, that is, the ability to escape from local optima by accepting sequences that momentarily deteriorate the objective function under specific condition.

### 3.2. Proposed SA-Based Approach

The main reason that we prefer to use SA instead of other metaheuristics is the fact that we can adapt our problem specific decisions and neighborhood structures to this approach rather than any other. For instance, genetic algorithm would not be as suitable as SA for this type of a problem since a mutation operator is hard to apply to the below defined solution vector. Similarly, tabu search would be difficult to implement as we would need more than one tabu list, one for each part of the solution vector. For this problem, the decisions to be made are the part sequence, machine assignments and the processing time allocation. Batur et al. observed that in such environments, it is usually enough to allocate only one part between the two machines [17]. Therefore, the solution is represented by a vector of length $2n + 2$, where $n$ represents the number of parts to be processed in the system. This vector consists of three parts, where the first part gives the sequence of the parts being processed, the second part shows the related machines that each part is to be processed and the last part gives the allocated part number together with its allocated processing time value. According to the sequence, assignment and allocations defined by this representation; all of the parts are taken from the input

buffer, transferred to related machines and delivered to the output buffer after their processes are completed.

### 3.2.1. SA Specific Decisions

In order to apply SA procedure to a combinatorial optimization problem, there are some decisions to be taken. Starting temperature must be hot enough to allow a move to almost any neighborhood state. However, if the temperature starts at a too high value then the search can move to any neighbor and thus transform the search (at least in the early stages) into a random search. In this study, four different starting temperature values (*50, 100, 150, 200*) are taken into account and it is observed that the best solutions are obtained by choosing the value to be $T_{init} = 100$.

It is usual to let the temperature decrease until it reaches zero. Some implementations keep decreasing the temperature until some other condition is met; for example, no change in the best state for a certain period of time. We assumed that the annealing process will continue until the temperature reaches 0, i.e. $T_{final} = 0$. The way in which we decrement our temperature is critical to the success of the algorithm. One way to decrement the temperature is a simple linear method. An alternative is a geometric decrement where $T = T * (DecRatio)$, where $DecRatio < 1$. Experience has shown that $DecRatio$ should be between *0.8* and *0.99*, with better results being found in the higher end of the range. In this study temperature decrement is determined taking $DecRatio = 0.99$.

Another decision we have to make is how many iterations we make at each temperature. A constant number of iterations at each temperature is an obvious scheme. We assumed the iteration number to be equal to the given number of parts of the considered problem.

### 3.2.2. Problem Specific Decisions

#### 3.2.2.1 Initial Solution

Initial solution for this problem is constructed randomly. Once the parts to be processed are sequenced in the first part of the above explained vector representation, second part of the solution is constructed being either the first or the second machines and finally one of the $n$ parts is defined along with its ratio of allocated time. We can explain this structure with an example of which there are 4 parts to be processed in the order of *3-4-1-2* on machines *2-1-1-2*, respectively and processing of the $4^{th}$ part is going to be shared by the two machines with a ratio of *30%* on the assigned machine and of *70%* on the other one. Such a solution is represented with the following vector: [3|4|1|2||2|1|1|2||4|0.3].

#### 3.2.2.2 Neighborhood Structure

In order to improve the current solution and get close to the optimum, algorithm searches for neighbor solutions. A neighborhood is defined by the set of feasible solutions that one can obtain with the help of swapping and random selections. There are four sub-strategies used in this study, and the one to use is determined randomly.

*Part Swapping:* First choice is to perform swap operation between parts. If we choose this way, we change the sequence of the two parts, given in the first part of the vector representation. For example, a neighbor solution of the vector [3|**4**|1|**2**||2|1|1|2||4|0.3] may be the vector [3|**2**|1|**4**||2|1|1|2||4|0.3], which has a different part sequence and also different machine assignments due to this sequence.

*Change of Assignment:* Second option is to change any one of the parts' machine assignment. Such a variation is obtained changing one point of the second part of the solution vector. For the above given example, the new vector [3|4|1|2||2|1|**1**|2||4|0.3] is possible to obtain, which means that part *1* is no longer processed by machine *2*, but its processing is performed by machine *1* instead.

*Change of Allocated Part or Its Allocation Ratio:* Third and fourth alternatives are related to the final part of the vector representation. If the third one is chosen then the part to allocate is changed, which is part *4* in the above example; whereas the last way is to change the allocated time ratio, which is *0,3* in the example.

#### 3.2.2.3 Objective Function Value

Throughout the solution procedure, presented with a solution to a problem, there exists some ways of measuring the quality of the solution. For this problem, there is only one objective to be minimized which is the total time that passes between the time that all of the parts are taken from the input buffer and that they all are left to the output buffer after being processed. Calculations of the objective function value are performed using the following steps.

At time zero, robot is assumed to be in front of the input buffer and all the parts are loaded in the buffer. Algorithm starts taking the first part of the sequence defined by the solution vector and loading it on the machine to which it is assigned. Afterwards, at any point of time, it is first checked whether or not any of the machines is empty. If the answer is positive, then the part to be loaded is determined according to the part sequence given in the solution vector and loaded on this machine. If both of the machines are loaded, robot will deal with the one of which the processing of the already loaded part is to be finished earlier. After unloading of this part, robot takes the next assigned one (if there are any left) from the input buffer and loads on this machine. When all of the parts of both machines are processed and carried to the output buffer, system returns to its initial position and the time that passed until this moment is taken as the objective function value. For the reader to get the insight of the algorithm better, we give a detailed representation of a 3-part example in Appendix A.

Proposed algorithm starts by constructing the above explained initial solution. After calculating the objective function value of this solution, four neighbors are developed with the help of the previously defined neighborhood structures. Objective function values are determined for each of these and the minimum valued solution is selected. This new solution is compared to the initial one and according to the SA approach, it is decided

whether to go on with this one instead of the previous. As it is commonly known, SA accepts the newly constructed solution in case of the objective function value is preferable, whereas it takes the probability function into account on the other case. In the same manner, in our algorithm, the best solution found in that iteration is compared with the one that this solution is obtained from and accepted if it is better or the generated random value fits the desired condition. In order to allow a worse solution to be taken as the next candidate, this state is integrated into the solution procedure defining a temperature value which is to be updated at each decision step. Flow-chart of the proposed approach is given in Fig. 2.

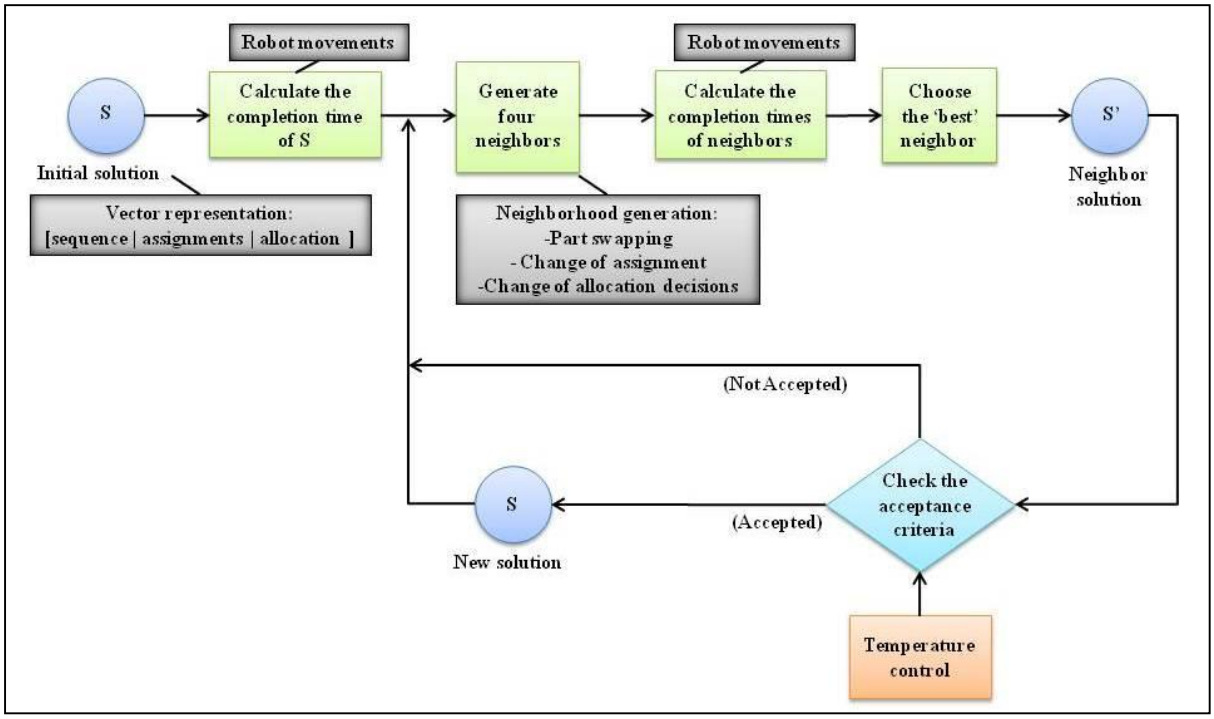



Fig.2. Structure of the proposed approach

Notations used in the algorithm are as follows and the pseudo-code of the algorithm is also given step-by-step in Fig.3.

| | |
|---|---|
| *n* | part number |
| *Sol* | initial solution |
| *SolObj* | objective function value of ´*Sol*` |
| *Current* | the solution which is used to obtain a new one |
| *CurObj* | objective function value of ´*Current*` |
| *IterSol* | neighbor solution in the iteration |
| *IterObj* | objective function value of ´*IterSol*` |
| *IterBestSol* | best solution found in the iteration |
| *IterBestObj* | objective function value of ´*IterBestSol*` |
| $T_{init}$ & $T_{final}$ | initial & final temperature |
| *Delta* | difference observed in the objective function values |
| *DecRatio* | cooling ratio |

**Step 1.** Produce the initial solution (*Sol*) randomly and compute its cost function value (*CurObj*).

Make *Current = Sol* and *CurObj = SolObj*.

**Step 2.** Set the parameters: $T_{init} = 100$, $T_{final} = 1$, *DecRatio = 0.99*.

Make the temperature of this moment equal to the initial temperature, i.e. $T = T_{init.}$.

**Step 3.** If $T > T_{final}$ go to Step 3.1, else go to Step 4.

**Step 3.1.** Make *IterBestSol = Ø* and *IterBestObj = ´a big number`*

Perform Step 3.1.1 to Step 3.1.2 for *n* times.

**Step 3.1.1.** Create a neighbor solution (*IterSol*) using either swap or random selection and compute its objective function value (*IterObj*).

**Step 3.1.2.** If *IterObj < IterBestObj*,

*IterBestSol = IterSol, IterBestObj = IterObj*.

**Step 3.2.** Compute the difference, *Delta = IterBestObj – CurObj*.

Define the random variable, *RanNum ~ U(0,1)*.

**Step 3.3.** If *IterBestObj < CurObj* or $e^{Delta/T} > RanNum$,

*Current = IterBestSol, CurObj = IterBestObj*.

**Step 3.4.** Update the temperature, *T = T*DecRatio* and go to Step 3.

**Step 4.** Save the final *Current* and *CurObj* values.

End the algorithm

Fig.3. Pseudo-code of the algorithm

## 4. COMPUTATIONAL EXPERIMENTS

In this study, we constructed an algorithm which involves an application of the simulated annealing algorithm. The method not only sequences the parts and the robot movements related to the production plan; but also enables the process allocation in order to help to balance the workload. In this manner, the method trying to minimize the makespan, satisfies effective solutions in short times.

Considered problem has some basic parameters; which are number of parts, average processing time, range of processing times, load / unload time and travel time [17]. In Table 1, these five factors which effects the complexity of the problems, and the levels related to these factors are given. These levels are constructed according to the observations obtained throughout the trials of the proposed algorithm.

Table 1. Factor and factor levels

| Factor | Low | High |
|---|---|---|
| Number of Parts | ≤ 30 | > 30 |
| Average Processing Time | ≤ 50 | > 50 |
| Range of Processing Times | ≤ 150 | > 150 |
| Load / Unload Time | ≤ 5 | > 5 |
| Travel Time | ≤ 5 | > 5 |

The algorithm of which the steps are defined in Section 3 is constructed and examined using Microsoft Visual C++ 2008. We have obtained satisfying solutions very quickly even for large sized problems. In order to have trustworthy comments over the quality of the obtained solutions, we have compared our method with LPT (Longest Processing Time) which is a commonly used algorithm. We use the Equation (1) in order to calculate the improvement.

$$\%\ deviation = \frac{C_{max}(LPT) - C_{max}(Heuristic)}{C_{max}(Heuristic)} * 100, \quad (1)$$

where $C_{max}(LPT)$ is the makespan obtained by the LPT approach and $C_{max}(Heuristic)$ is the one obtained by our algorithm.

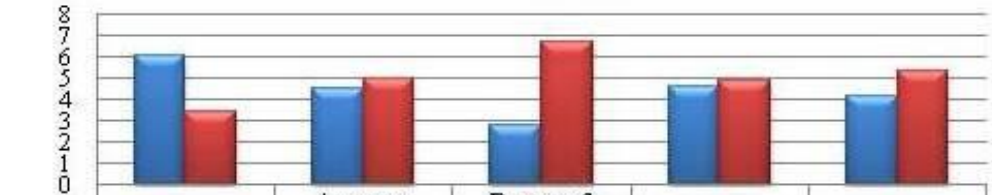
We have run both of the algorithms for *30* times for all combinations of the above five factors and observed that the proposed algorithm gives very good results compared to the ones of LPT. Only *17* of a total of *960* runs, LPT results are better with an average improvement of *0,20 %*. For the rest, SA based algorithm is superior, giving an average improvement of *4,75 %.* As is previously mentioned, this problem type was considered by Batur et al. [17] in a cyclic manner. They also compared the results with LPT and obtained an improvement of *2,67 %* over this method. Considering the production plan to be non-cyclic, we have compared the results with the same algorithm and by the help of SA procedure, we have increased this gain by more than *2 %*, without any extra effort.

Table 2. Improvement percentages

| Factor | Factor Level | Mean (%) | Minimum (%) | Maximum (%) |
|---|---|---|---|---|
| Number of Parts | Low | 6,09 | -1,47 | 19,92 |
| | High | 3,41 | -0,35 | 17,89 |
| Average Processing Time | Low | 4,53 | -0,18 | 18,70 |
| | High | 4,97 | -1,47 | 19,92 |
| Range of Processing Times | Low | 2,80 | -0,35 | 19,92 |
| | High | 6,71 | -1,47 | 19,67 |
| Load / Unload Time | Low | 4,60 | -1,47 | 19,67 |
| | High | 4,90 | -0,24 | 19,92 |
| Travel Time | Low | 4,20 | -0,35 | 19,54 |
| | High | 5,30 | -1,47 | 19,92 |

Average, minimum and maximum improvements obtained by the proposed method are given in Table 2. As can be observed from the table, our algorithm has satisfied considerably high improvements like *19,92 %*, and in worst cases, its deterioration is only *1,47 %.*

In order to take a closer look to the factor levels separately, we use Fig. 4. It is observed that average processing time and load / unload times are not as effective as the other three factors. Advantages of the proposed algorithm seems to be more obvious for cases with low number of parts, high ranged processing times and high valued travel times. These observations are clearly reasonable. Starting from the first factor, LPT's disadvantages gets less observable as the number of parts gets higher. Increment over the range of processing times means that there are various types of parts to be produced and it is not reasonable to schedule these items by the help of an algorithm which is insensitive to these differences. Finally, travel time is important as we consider robotic cells and it is related to the most significant parameter, the robot movements, that needs to be decreased.




| | # Parts | Average Processing Time | Range of Processing Times | Load/Unload Time | Travel Time |
|---|---|---|---|---|---|
| Low | 6,09 | 4,53 | 2,8 | 4,6 | 4,2 |
| High | 3,41 | 4,97 | 6,71 | 4,9 | 5,3 |

Fig. 4. Average improvement percentages according to the factor levels

It is previously mentioned that we let part allocations throughout this study. As can be seen from Table 3, in *86* out of *960* runs, the algorithm chose to use this option. The result responses to an amount of *8,96 %* in total, and

shows that allocation is able to be a preferable option again without the need of any additional sources. Table 3 also shows the allocation numbers related to the previously mentioned factor levels. We can say the results are fairly distributed among the levels of all but only one factor. The reason that load / unload time to be the most effective parameter of the decision of allocation is the necessity of the allocated part to be loaded and unloaded twice. One can easily say that it is more possible the allocation option to give better results as this factor takes lower values.

Table 3. Allocation numbers

| Factor | Factor Level | Number |
|---|---|---|
| Number of Parts | Low | 47 |
| | High | 39 |
| Average Processing Time | Low | 45 |
| | High | 41 |
| Range of Processing Times | Low | 48 |
| | High | 38 |
| Load / Unload Time | Low | 64 |
| | High | 22 |
| Travel Time | Low | 42 |
| | High | 44 |
| Total | | 86 |

## 5. CONCLUSION

In this study, we focus on the scheduling problem arising in two-machine flexible robotic cells which can process lots that contain different types of parts. Considering multiple part types, the decisions to be made are to find the robot move sequence and the part sequence, that jointly minimize the total production time. As we do not assume the allocation of processing times on each machine to be constant, we also have the processing time allocation problem.

Due to the fact that mathematical optimization techniques are very limited to use for NP-hard problems, in the existing literature, it is very common to use metaheuristic approaches solving this type of 'hard to solve' optimization problems. We have proposed an efficient Simulated Annealing Algorithm which is also very easy to implement and can be extended for further cases with ease. The improvement search of the algorithm uses both the swapping and random selections. Experimental results show that it is possible to get acceptable solutions very quickly and effectively. It is important to notice that we obtained better results only by changing processing time allocations for one part; with no additional effort or cost. As a future research direction, although explained procedure is developed for a *2*-machine system, it is not hard to expand the algorithm for *3*- or *m*-machine cases.

## REFERENCES

[1] Dawande, M., Geismar, N., Sethi, S.P., "Dominance of cyclic solutions and challenges in the scheduling of robotic cells", SIAM Review, 47, 709-721, (2005).

[2] Gundogdu, E., Gultekin, H., "Scheduling in two-machine robotic cells with a self-buffered robot", IIE Transactions, 48(2): 170-191, (2016).

[3] Gan, H-S., Wirth, A., Abdekhodaee, A., "A branch-and-price algorithm for the general case of scheduling paralel machines with a single server", Computers and Operations Research, 39: 2242-2247, (2012).

[4] Hasani, K., Kravchenko, S.A., Werner, F. "Minimising interference for scheduling two parallel machines with a single server", International Journal of Production Research, 52(24): 7148-7158, (2014).

[5] Jiang, Y., Zhang, Q., Hu, J., Dong, J., Ji, M., "Single-server parallel-machine scheduling with loading and unloading times", Journal of Combinatorial Optimization, 30(2): 201-213, (2015).

[6] Hasani, K., Kravchenko, S.A., Werner, F., "Minimizing the makespan for the two-machine scheduling problem with a single server: Two algorithms for very large instances", Engineering Optimization, 48(1): 173-183, (2016).

[7] Balin, S., "Nonidentical parallel machine scheduling using genetic algorithm", Expert Systems with Applications, 38, 6814-6821, (2011).

[8] Bilyk, A, Mönch, L., "A variable neighborhood search approach for planning and scheduling of jobs on unrelated parallel machines", Journal of Intelligent Manufacturing, 23(5), 1621-1635, (2012).

[9] Browne, J., Harhen, J., Shivnan, J., Production Management Systems. New York: Addison-Wesley, (1996).

[10] Gultekin, H., Akturk, M.S., Karasan, O.E., "Robotic cell scheduling with tooling constraints", European Journal of Operational Research, 174, 777-796, (2006).

[11] Jolai, F., Foumani, M., Tavakoli-Moghadam, R, Fattahi, P., "Cyclic scheduling of a robotic flexible cell with load lock and swap", Journal of Intelligent Manufacturing 23(5), 1885-1891, (2012).

[12] Mansouri, S.A., Hendizadeh, S.H, Salmasi, N., "Bicriteria scheduling of a two-machine flowshop with sequence-dependent setup times", International Journal of Advanced Manufacturing Technology, 40, 1216-1226, (2009).

[13] Naderi, B., Fatemi Ghomi S.M.T., Aminnayeri, M., "A high performing metaheuristic for job shop scheduling with sequence-dependent setup times", Applied Soft Computing, 10, 703-710, (2010).

[14] Werner, F., Kravchenko, S.A., "Scheduling with multiple servers". Automation and Remote Control. 71, 2109-2121, (2010).

[15] Mor, B., Mosheiov, G., "Heuristics for scheduling problems with an unavailability constraint and position-dependent processing times", Computers and Industrial Engineering, 62, 908-916, (2012).

[16] Jiang, Y., Dong, J., Ji, M., "Preemptive scheduling on two parallel machines with a single server", Computers and Industrial Engineering, 66, 514-518, (2013).

[17] Batur, G.D., Karasan, O.E., Akturk, M.S., "Multiple part-type scheduling in flexible robotic cells", International Journal of Production Economics, 135, 726-740, (2012).

[18] Metropolis, N., Rosenbluth, A., Resenbluth, M., "Equation of state calculations by fast computing machines", Journal of Chemical Physics, 21, 1087–1092, (1953).

[19] Kirkpatrick, S., Gelatt, Jr., C.D., Vecchi, M.P., "Optimization by simulated annealing", Science, 220, 671-680, (1983).

[20] Hooda, N., Dhingra, A.K., "Flow shop scheduling using simulated annealing: A review", International Journal of Applied Engineering Research, 2, 234-249, (2011).

[21] Low, C., Yeh, J.Y., Huang K.I., "A robust simulated annealing heuristic for flow shop scheduling problems", International Journal of Advanced Manufacturing Technology, 23, 762-767, (2004).

[22] Lee, Z.J., Lin, A.W., Ying, K.C., "Scheduling jobs on dynamic parallel machines with sequence-dependent setup times", International Journal of Advanced Manufacturing Technology, 47, 773-781, (2010).

[23] Behnamian, J. Zandieh, M., Fatemi Ghomi, S.M.T., "Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm", Expert Systems with Applications, 36, 9637-9644, (2009).

[24] Zhang, R., Wu C., "A hybrid immune simulated annealing algorithm for the job shop scheduling problem", Applied Soft Computing, 10, 79–89, (2010).

## APPENDIX A. NUMERICAL EXAMPLE

We use the following example: Assume that we have 3 parts to be completed with corresponding processing times: $P_1 = 6$, $P_2 = 12$ and $P_3 = 40$, ε and δ are both 1 units of time and solution of $[3|1|2||1|2|1||3|0.4]$ is obtained throughout the solution procedure. This solution corresponds to the situation that part *2* being processed on the *1st* machine, part *1* on the *2nd* machine and part *3* is to be allocated as *0.4* of its total processing is going to be performed by the *1st* machine and the rest on the other.

As it is mentioned before, at the starting moment ($time(t) = 0$) robot is in front of the input buffer, all the parts are loaded in the input buffer, both of the two machines are empty and the decision of which machine to load first is made according to the sequence given in the first part of the solution vector. In such a situation, robot starts its movement taking the *3rd* part (allocated one) to the *1st* machine. Such a movement necessitates robot taking the part from the input buffer (ε), carrying it to the machine (δ) and loading (ε). Robot time is $t = 2\varepsilon + \delta = 3$ now. According to the solution vector, processing of this part is shared among the two machines as $P_{31} = 40 * 0.4 = 16$ and $P_{32} = 40 * 0.6 = 24$, so its processing on the *1st* machine will be completed at time $t = 3 + 16 = 19$. For the following step, since the *2nd* machine is empty and its next part (part *1*) has not been loaded yet, robot performs the movement related to this part; goes back to the input buffer first (δ), takes the part (ε), brings it to the *2nd* machine (2δ), and loads the part on the machine (ε). This movement makes the time $t = 3 + 2\varepsilon + 3\delta = 8$. Now, both of the machines are loaded. Completion times of parts *3* and *1* are *19* and *14*, respectively. For the following movement, *1st* machine is selected as its processing is completed earlier. Since robot is still in front of the machine, it does perform any travel and waits until the processing is completed. At time $t = 14$, robot takes part *1* from the *2nd* machine (ε) and brings it to the output buffer (δ). Now there is one part in the system and one waiting in the input buffer to be processed. As the next part is part *2* and it is assigned to the *1st* machine, this machine needs to be unloaded first. At this machine part *3* is loaded and its next location is the *2nd* machine which is already unloaded, so robot goes to the *1st* machine from the output buffer (2δ), does not wait as its processing is completed at that time ($t = 19$), takes the part (ε), brings (δ) and loads it on the *2nd* machine (ε) at time $t = 22$. This part needs to be processed at this machine for *24* units of time. *1st* machine is now ready for the next machine to be loaded. Robot goes back to the input buffer from the *2nd* machine (*2*δ), takes the last part which is part *2* (ε), carries to the machine (δ) and loads it on the *1st* machine (ε) at time $= 27$ . Now all of the parts of both of the machines are loaded once. In order the system to return back to its initial position, both of the machines needs to be unloaded. As the completion time of part *2* is smaller than the other one, robot will wait in front of the *1st* machine until time $t = 39$, unloads it (ε+*2*δ+ ε) and goes to the *2nd* machine for part *3* (δ). At time $t = 46$, processing of the last part is completed and it is taken to the output buffer as soon as it is unloaded from the machine (ε+δ+ ε). As all of the parts' processes are completed, robot goes to its initial position (input buffer) (3δ) which makes the system return to its initial state. At this point of time, the time passed until this moment is calculated to be $t = 52$ which is defined as the objective function value.

Related Gantt-chart is given by Fig. A.1.

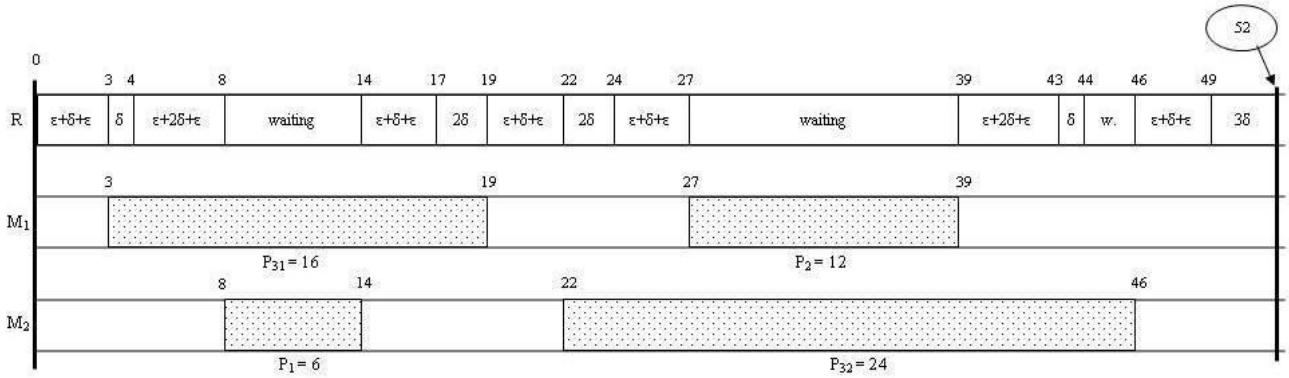



Fig. A.1. Gantt-chart for the example