ELSEVIER

Robotics and Computer-Integrated Manufacturing 20 (2004) 143–150

Robotics and Computer-Integrated Manufacturing

www.elsevier.com/locate/rcim

Minimum makespan task sequencing with multiple shared resources

Massimiliano Caramia[a,*], Paolo Dell'Olmo[b], Riccardo Onori[c]

[a] *Istituto per le Applicazioni del Calcolo IAC-CNR "Mauro Picone", Viale del Policlinico, 137 - 00161 Rome, Italy*

[b] *Dipartimento di Statistica Probabilità e Statistiche Applicate, Università di Roma "La Sapienza", Piazzale Aldo Moro, 5 - 00185 Rome, Italy*

[c] *Dipartimento di Informatica, Sistemi e Produzione, University of Rome "Tor Vergata", Via del Politecnico, 1 - 00133 Rome, Italy*

Abstract

In this paper we study the general problem of sequencing multiple jobs, where each job consists of multiple ordered tasks and tasks execution requires simultaneous usage of several resources. In particular, the case of an automatic assembly cell is examined. NP-completeness results are given. A heuristic is designed and evaluated.

MSC: 68M20; 68Q17; 68Q25; 68W40; 90B35; 90C27

Keywords: Scheduling; Robotic cells; Job shop heuristic algorithm; Complexity analysis

1. Introduction

A substantial amount of recent research has been directed towards the development of industrial robots. The bulk of this work has dealt with electromechanical capabilities, sensing devices and computers controls. Relatively little research has investigated the operational problem associated with application of this technology [1–5]. We investigate one operational problem which is encountered in applications, in which a robot is used to tend a number of machines. Such an application would arise, e.g., when machines have been organized into a machine cell to implement the concept of group technology [6–11]. The cell would be used to produce multiple set of parts at prespecified production rates. The feasibility of assigning one robot to perform the tasks necessary for tending all the machines, so that parts are produced at specified production rates, is an important operational problem. In fact, the resolution of this problem determines the number of robots that might be necessary to tend machines in a manufacturing system and hence the investment required to robotize tending activities.

These kinds of problems were first introduced by Asfahl [12]. A summary of the related literature can be found in [13,14]. In particular, in [13] the authors find the optimal sequence of moves for a two-machine robot-centred cell producing a single part-type, and solve the part sequencing problem for a given one-unit robot move cycle in a two-machine cell producing multiple part-types. Hall et al. [14] showed that the optimal solution to the multiple part-types problem in a two-machine cell is not generally given by a one-unit cycle. They develop an $O(n^4)$, where n is the number of parts, time algorithm that jointly optimizes the robot move cycle and part sequencing problems. The latter algorithm was improved later by Aneja and Kamoun [2] with one of $O(n \log n)$ complexity.

Also, Blazewicz et al. [15] provided a summary of the related literature and described a line for machining castings for truck differential assemblies in the form of a three-machine robotic cell where a robot has to transfer heavy mechanical parts between large machines.

Descriptions of interesting applications were provided, e.g. by Hartley [16]. Complexity issues were discussed by Wilhelm [17], and Crama and Van De Klundert [18], that provided a proof that a basic version of the problem is strongly NP-complete, and described a polynomial time algorithm for minimizing cycle time over all one-unit cycles in an m-machine cell producing a single part-type in a robot-centred cell.

*Corresponding author. Fax: +39-06-44-04-306.

E-mail addresses: caramia@iac.rm.cnr.it (M. Caramia), paolo.dellolmo@uniroma1.it (P. Dell'Olmo), onori@disp.uniroma2.it (R. Onori).

0736-5845/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rcim.2003.10.003

Hall et al. in [4] considered a three-machine cell producing multiple part-types, and proved that, in two out of the six potentially optimal robot move cycles for producing one unit, the recognition version of the part sequencing problem is unary NP-complete. Moreover, they have shown that the general part sequencing problem not restricted to any robot move cycle in a three-machine cell is still intractable.

Levner et al. [19] addressed a cyclic robot scheduling problem in an automated manufacturing line in which a single robot is used to move parts from one workstation to another, with the goal of optimizing the cycle length. For this problem they proposed an algorithm of complexity $O(m^3 \log m)$ where m is the number of machines.

Many of the results and algorithms in the literature are devoted to robotic flow-shop scheduling problems (e.g., see [12,14,15,20]). However, the robotic cell configuration is very flexible: the robot can easily access the machines in any order, thus producing a large variety of products in the form of a job-shop (e.g., see [21–24]).

In this paper we concentrate on the latter class of problems, studying the general problem of sequencing multiple jobs where each job consists of multiple ordered tasks and tasks execution requires simultaneous usage of several resources [25].

The remainder of the paper is organized as follows. In Section 2 we formally describe the problem. The complexity of the problem is analyzed in Section 3. A heuristic algorithm is described in Section 4, and finally in Section 5 we present computational results.

2. Problem definition

Let us consider a cell composed of M machines configured to produce a batch of parts. Part p, $p = 1, \ldots, P$, requires N_p tasks. Let p_j denote the jth task for part p. Task p_j takes t_{p_j} time units to complete. We allow the general condition that a task requires concurrent usage of more than one cell resource during its execution. Let S_{p_j} be the set of resources required for task p_j. The objective is to determine the schedule for all tasks of all parts so as to minimize the makespan. Note that the mainstream of research in robotic scheduling is devoted to two classes of production performance measures. The first is the makespan which addresses the case of a finite part set where one is interested in minimizing the maximum completion time of these parts (e.g., see [23,26–28]). The other class of models (which is not addressed in this paper) assumes that the part set is infinite and attempts to minimize the long run cycle time of the schedule, which is the same as maximizing the throughput rate (e.g., see [13]).

As an example of this model, we examine an assembly cell of $M-1$ machines plus a material handling robot. The robot will be modelled as machine M. Each part has a processing time associated with each machine. In addition, the robot is needed to load a part on a machine and then to unload the part after the production task has been performed. In our model, each production task is divided into three tasks, namely Load, Make, and Unload. The Load and Unload tasks require both the machine and the robot. The Make operation would normally require only the machine. The formulation also permits the case where the robot is needed to hold the part while certain production tasks are performed. This problem can be easily extended to the case of multiple material handlers, each assigned to a specific set of machines. Note that the solution must take into account precedence restrictions. For instance, we cannot unload until we have loaded and performed the Make task.

This is an example of operational problems associated with the application of industrial robots, used to tend a number of machines that have been organized into a machine cell. The cell is used to produce a set of parts at production rates specified by managers. The time required to tend a machine, may be different for each machine according to its location and orientation in the cell, and to the part to be processed. All machines in the cell are dependent on a single robot, so the sequence in which tending tasks are performed may be critical and force certain machines to be in idle.

Let us denote the Load, Make and Unload operations of part p requiring machine i with L_p^i, M_p^i, U_p^i (the machine index i will be omitted when not necessary). Moreover, we shall refer to the problem of scheduling Load, Make and Unload operations for the set of parts in order to minimize makespan as the LMU problem.

As an example of LMU problem, in Fig. 1 is shown a schedule, for three parts and two making machines. In particular, the robot is indicated with R, and the machines with m_1 and m_2, respectively. Moreover, the Make operation for part 2, namely M_2^2, requires resource R.

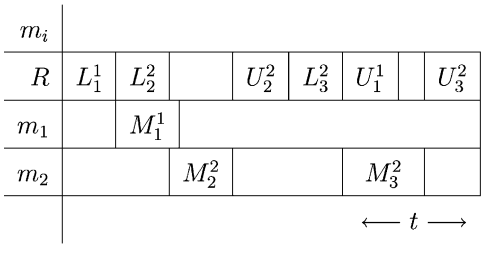

Fig. 1. An example of LMU problem.

3. NP-completeness result

We show that *LMU* problem with general processing times on four machines is NP-hard by a transformation from 3-Partition, which is known to be strongly NP-complete (see [29]), to our problem. The 3-Partition problem is defined as follows.

3-*Partition problem*: Given a set $A = \{a_1, a_2, \ldots, a_{3z}\}$ of $3z$ integers such that $\sum_{i=1}^{3z} a_i = zB$ and $B/4 < a_i < B/2$ for $i = 1, \ldots, 3z$, can A be partitioned into z disjoint subsets, $A_1, A_2, \ldots, A_z$, such that $\sum_{a_i \in A_k} a_i = B$ for each $k = 1, 2, \ldots, z$?

Theorem 1. *The LMU problem with* $m = 4$ *is strongly* NP-*complete*.

Proof. For a given instance of the 3-Partition problem let us define a corresponding instance of our problem with $5z$ parts. Let there be $2z$ parts requiring machine 1, z parts requiring machine 2, z parts requiring machine 3, and z parts requiring machine 4. Recalling the definition of B in the statement of the 3-Partition problem, let the processing times be as follows:

- $L_p^1 = 1$, $M_p^1 = B$, $U_p^1 = 1$, $p = 1, \ldots, 2z$;
- $L_p^2 = a_{p-2z}$, $M_p^2 = B - a_{p-2z} + 2$, $U_p^2 = a_{p-2z}$, $p = 2z + 1, \ldots, 3z$;
- $L_p^3 = a_{p-3z}$, $M_p^3 = B - a_{p-3z} + 2$, $U_p^3 = a_{p-3z}$, $p = 3z + 1, \ldots, 4z$;
- $L_p^4 = a_{p-4z}$, $M_p^4 = B - a_{p-4z} + 2$, $U_p^4 = a_{p-4z}$, $p = 4z + 1, \ldots, 5z$.

If the 3-Partition problem has a positive answer, then we can construct a schedule of length $Y = 2z(B + 2)$ as shown in Fig. 2.

Now, we are going to show that there is a positive answer to 3-Partition if there exists a feasible schedule with length less than or equal to Y. We observe that the value of the makespan to schedule in any order the $2z$ parts requiring machine 1 is $Y = 2z(B + 2)$. This partial schedule has no idle times on machine 1 and has $2z$ idle times on the robot all of length B. The total time required to perform the Load and Unload operations of the remaining $3z$ parts is $2zB$. Note that a feasible schedule for the remaining parts can be obtained

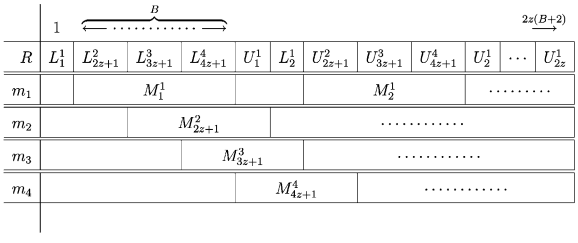

Fig. 2. An optimal schedule with $Y = 2z(B + 2)$.

scheduling in any order the Load operations of parts on machines 2–4, on an idle time of the robot. The Make operation can start as soon as the Load is completed. Unload operations can be performed in the successive robot idle time in the same order than the corresponding Load operations. In a schedule of length Y the robot must have no idle times. This is possible if the sum of the Load (Unload) operations of parts requiring machines 2–4 in each robot idle time is equal to B, that is if it exists a 3-Partition of the set A. Thus the problem is strongly NP-complete. □

4. The heuristic

In the following, we describe a heuristic algorithm (namely *LMUA*) which finds a feasible solution to our scheduling problem.

First, we observe that an active schedule can have idle times either on one of the machines or on the robot which cannot be eliminated trivially. The robot can be idle when all machines are simultaneously processing a Make task. A machine m_i may be idle, waiting for either a Load or an Unload task to be performed, because the robot is busy tending another machine. Second, for any feasible schedule the maximum completion time is the completion time of the last Unload operation.

The *LMUA* algorithm proposed is a single pass heuristic in which the loading–unloading sequence and the corresponding schedule are determined once. A list containing the sequence of Load–Make–Unload tasks is built considering any order of the part types. At the beginning the robot R loads all machines. Make operations can start immediately after the preceding Load is performed. Successively the robot unloads the machine which ended first the Make task.

In the generic stage of the algorithm the first unselected task in the list is examined. If it is a Make operation it can be scheduled immediately after the loading. Otherwise, the first Load–Unload operation in the list of remaining tasks which tends the machine which has been idle for the longest time is selected. The following is a pseudo-code description of algorithm *LMUA*.

Algorithm LMUA. *Step* 1: Consider an instance with $M - 1$ machines, one robot (modelled as machine M) and Z parts.

1.1. Take any ordering of all the parts (assume the order $1, \ldots, Z$);
1.2. Build the list of tasks: $LT = \{L_1^k, M_1^k, U_1^k, L_2^k, M_2^k, \ldots, L_Z^k, M_Z^k, U_Z^k\}$;
1.3. Build the list of tasks Make that require the resource robot: $LTR = \{M_i^k |$ part i requires the robot during the Make on $m_k\}$;

1.4. Build the list of processing times: $PT = \{p_{L_1^k}, p_{M_1^k}, p_{U_1^k}, \ldots, p_{U_Z^k}\}$;
1.5. Build the list of the instants of time at which the machines are available: $AT = \{At_1, At_2, \ldots, At_M\}$;
1.6. Initialize the current scheduling time at $t = 0$;
1.7. Initialize the list of the tasks that can be processed at the current t with all the Load tasks: $LTA = \{L_1^k, L_2^k, \ldots, L_Z^k\}$;
1.8. Build the list reporting the instants of time at which the tasks in list LT could start their execution: $FTA = \{Ft_{M_1^k}, Ft_{U_1^k}, \ldots, Ft_{U_Z^k}\}$;
1.9. Set the values of the variables in FTA equal to infinite.

Step 2: While $LT \neq \emptyset$ or $LTA \neq \emptyset$ do:

2.1. Scan tasks in list LTA and
  2.1.1. if there exists a task Make that at time t requires a machine m_k that is available according to list AT then go to Step 3; otherwise
  2.1.2. if the robot is available at time t and there exists either a task Load whose corresponding Make operation requires a machine which is available, or a task Unload, then go to Step 4 (tie breaks choosing the Load or Unload task waiting for more time in the list).
2.2. If there does not exist a task obeying Step 2.1.2 or Step 2.1.3, then:
  2.2.1. Increase $t = t + 1$;
  2.2.2. Update lists LTA and LT, by moving tasks from LT to LTA according to FTA.

Step 3:

3.1. Schedule the Make task selected starting from t on the required machine;
3.2. Set equal to infinite the variable in AT associated with the machine handling the Make task.
3.3. Update in FTA the earlier starting time of the Unload task associated with the processed Make task, setting it to the finishing time of the latter;
3.4. Delete from LTA the processed Make task.
3.5. Set $t = t + 1$;
3.6. Go to Step 2.

Step 4: If the selected task is a Load (whose Make task does not require the robot) or an Unload, then:

4.1. Process the task;
4.2. Update the instant of time At_M at which the robot will be available again according to the processing time of the task executed; set t to this latter value.
4.3. If a Load task has been selected, then update in FTA the earlier starting time of the Make task associated;
4.4. If an Unload task has been selected, set to $t + 1$ the time at which the machine which have processed its predecessor Make task will be available; update the instant of time At_M at which the robot will be available again according to the processing time of the task executed; update t accordingly.
4.5. Delete from LTA the processed task;
4.6. Go to Step 2.

If the selected task is a Load task such that the following Make task requires the presence of the robot (as shown in list LTR), then:

4.7. Process the task Load and immediately after the following task Make;
4.8. Update the variable in list AT indicating when the robot will be available again (i.e., after an interval of time equal to the sum of the processing times of the Load and the Make operations), while set the availability of the machine which has processed the Make task equal to infinity after this Make operation has been performed;
4.9. Update the variable in list FAT of the earlier starting time of the corresponding Unload task, i.e., t plus the processing times of the Load and the Make operations performed, say $p_{L_i^k}$ and $p_{M_i^k}$, respectively;
4.10. Update $t = t + p_{L_i^k} + p_{M_i^k}$;
4.11. Delete from LTA the Load task;
4.12. Delete from LT the Make operation.
4.13. Go to Step 2.

Step 5: Return the makespan: $C_{\max} := t$

In Fig. 3 we show an application of *LMUA* algorithm with four parts: Make operations for parts 1 and 3 require machine m_2; Make operations for parts 2 and 4

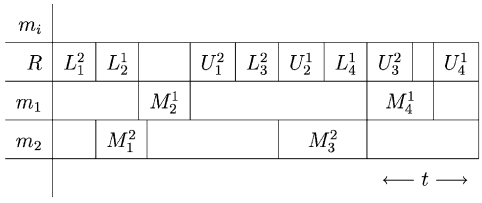

Fig. 3. An application of *LMUA* algorithm.

require machine m_1; moreover, Make operation for part 2, namely M_2^1, also requires resource R.

In order to determine the computational complexity of the *LMUA* algorithm, note that, for a given instance of Z parts, there are $3Z$ tasks and:

Step 1: the lists can be constructed in $O(Z)$;

Step 2: the cycle **while** is repeated at most $3Z$ times, and the task selection requires at most $3Z$ comparisons;

Step 3: runs in $O(1)$;

Step 4: can be processed in $O(1)$;

Step 5: runs in $O(1)$;

Hence, for a given instance of the *LMU* problem, denoting with Z the size of the input, *LMUA* algorithm has a worst case complexity $O(Z^2)$.

5. Computational results

In this section we present some computational experiments with *LMUA* algorithm on randomly generated problems. We considered several cell configurations with $m \in [4, 10]$ machines and one robot tending all the machines. For each configuration we considered an increasing number of jobs $n \in [10, 80]$. Note that, for instance, 80 jobs correspond to 240 tasks. Processing times for each loading and unloading operations are generated randomly, using a uniform distribution in the range [20–70] time units. Processing times of Make operations are generated randomly using a uniform distribution in the range [120–360] time units. We considered different scenarios associated with a probability p_r that a generic part requires the robot during the Make operation equal to 0, 0.1, 0.2, 0.3, and 0.4. The algorithm implementation was done in WINDOWS/C environment on a AMD Athlon PC running at 900 MHz.

Results are summarized in the following tables, which reports, for each cell configuration, the values of the makespan depending on the number n of jobs. Each table is associated with a scenario corresponding to a probability p_r.

First, let us consider a scenario in which jobs do not require the resource robot during the Make operation (Table 1). Observe that, for a given m, the makespan increases as n increases, while it decreases, for a given n, as the number of machines m increases. To evaluate the behaviour of the makespan we report the chart in Fig. 4.

The makespan value, given a fixed number m of machines, seems to be linearly dependent on the number of jobs. For a certain range of the number of jobs ($n \in [10, 30]$), the trends are very similar, and it seems that the number m of machines does not affect $C_{\max}$. As n increases, instead, the trends are well distinguished, and the difference between the makespan values is much more observable when m passes from 4 to 6, than when it passes from 6 to 8 or 10.

Now we analyze the behaviour of the makespan referring to the increase of the number m of the machines, for a given n. Fig. 5 shows how $C_{\max}$ decreases proportionally as the number m increases. Moreover, from Table 1 it is easy to see that when the number of jobs passes from $n = 30$ to 40 the variation of the associated makespan values is higher than in the other cases.

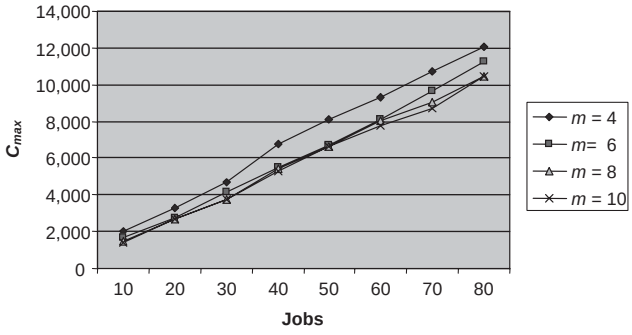

Fig. 4. Trends of the makespan as the number n of jobs increases.

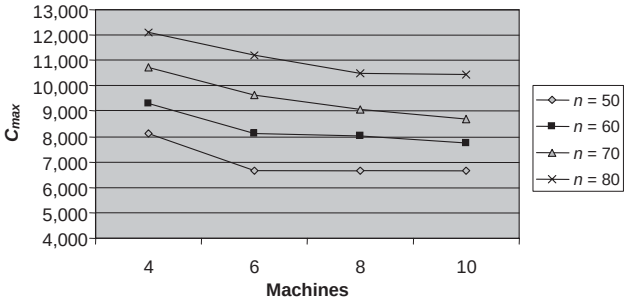

Fig. 5. Trends of the makespan as the number m of machines increases.

Table 1
Scenario with $p_r = 0$

$p_r = 0$	← Jobs (n) →							
Mac. (m)	10	20	30	40	50	60	70	80
4	1977.9	3265.5	4655.9	6774.6	8110.6	9300.4	10711.1	12077.3
6	1681.8	2733.4	4157.2	5460.4	6673.2	8117.8	9633.8	11220.8
8	1460	2712.1	3784.2	5400.4	6660	8014	9054.3	10482
10	1438.1	2678.2	3723.2	5314.5	6638.7	7754.5	8698.9	10458.3

Table 2
Scenario with $p_r = 0.1$

$p_r = 0.1$	← Jobs (n) →							
Mac. (m)	10	20	30	40	50	60	70	80
4	2069.9	3486.6	5118.2	7300	8923.4	9999.7	11690.6	13615.2
6	1747.8	3096.8	4536	6292.1	7830.5	9298.7	11094.8	13057.5
8	1534.3	3077.5	4487	6193.8	7510.2	8952.5	10382.5	12008.9
10	1506.4	2969.3	4322	6125.6	7503.8	8771.4	10138.7	11849.9

Table 3
Scenario with $p_r = 0.2$

$p_r = 0.2$	← Jobs (n) →							
Mac. (m)	10	20	30	40	50	60	70	80
4	2152.5	3674.4	5320.9	7808.6	9428.6	10962.4	12574.6	14435.3
6	1818.3	3458.4	5143.6	7109.3	8806.4	10647.6	12018.1	14319
8	1775.3	3373.7	4954	6761.6	8430.1	10336.6	11880	13752
10	1591.9	3367.3	4919.1	6651.7	8283.1	9914.6	11270.6	13356.7

Table 4
Scenario with $p_r = 0.3$

$p_r = 0.3$	← Jobs (n) →							
Mac. (m)	10	20	30	40	50	60	70	80
4	2301.7	3743.1	5611.7	8289	10223	11656.9	13882.7	15894.9
6	1929.2	3644.8	5462.2	7798.4	9358.6	11496.9	13212	15787.6
8	1821	3465.1	5440.8	7341.7	9207.9	10969.7	12786.3	15046.8
10	1760.1	3412.8	5092	7252.4	9048.8	10923.3	12089.6	14451

Table 5
Scenario with $p_r = 0.4$

$p_r = 0.4$	← Jobs (n) →							
Mac. (m)	10	20	30	40	50	60	70	80
4	2383	4134	6073.6	9009.7	10859.4	12837.6	14857.8	17176.5
6	2024.3	3960.9	5876.7	8243.3	10232.2	12491.2	14268.5	16624.1
8	1837.3	3671.6	5734.9	7969.9	9644.2	11775	13719.8	15950.3
10	1815.5	3603.2	5406.4	7700.9	9577.2	11424.9	13693.4	15296

Finally, we study what happens if the probability p_r that a job requires the robot during the Make operation is greater than zero. Tables 2–5 summarize such results.

Note that as p_r increases $C_{\max}$ decreases proportionally. The chart in Fig. 6 shows that the makespan values increase proportionally with the probability p_r and the number of jobs n.

The chart in Fig. 7, instead, shows that the variation of the makespan, when the probability p_r increases, is not proportional to the number m of machines, for a given n. In fact, it can be seen how the influence of p_r on the makespan tends to decrease as m increases.

Note that the maximum CPU time (in seconds) spent by the algorithm was 2.67, found for the combination $(p_r = 0.4, m = 6, n = 80)$, whereas the average running time was 0.38.

A final analysis is devoted to the possible efficiency improvement of the robotic cell. In particular, we examine whether is more profitable to reduce the processing times of Make operations, improving the machines efficiency, or to reduce the processing times for loading and unloading operations, modifying the robot configuration.

First we analyze, in the case $p_r = 0$, what happens if the range of the uniform distribution for the processing

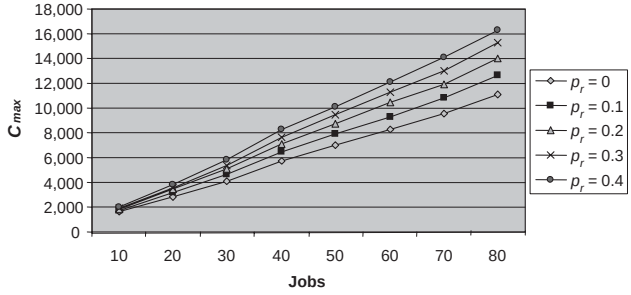

Fig. 6. Makespan values, for a given m, as n and p_r increase.

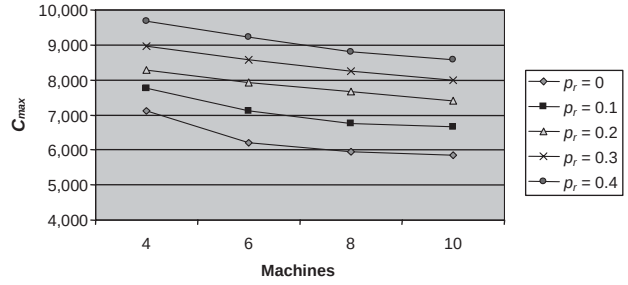

Fig. 7. Makespan values, for a given n, as m and p_r increase.

times of Make operations is decreased of 30%, i.e., from [120 − 360] to [120–268] time units. Table 6 summarizes the results obtained in this new scenario, and Table 7 reports the percentage of reduction of $C_{\max}$.

Now we analyze what happens if the range of the uniform distribution for the processing times of Load and Unload operations is decreased of 30%, i.e., from [20–70] to [20–55] time units. Table 8 summarizes the results obtained in this case, and Table 9 reports the percentage of reduction of $C_{\max}$.

It is easy to observe that a decrease of the processing times in both cases brings a reduction of $C_{\max}$ even if the latter is not proportional to the former. In fact, the maximum reduction obtained is 19.89% when processing times of loading and unloading operations are decreased, and 19.10% when processing times of Make operations are decreased instead.

6. Conclusions

We studied the general problem of sequencing multiple jobs where each job consists of multiple ordered

Table 6
Reducing processing times of Make operations

$p_r = 0$	← Jobs (n) →							
Mac. (m)	10	20	30	40	50	60	70	80
4	1600.1	2781.3	3831.6	5735.3	6806.3	7877.1	8862.9	10284.7
6	1412.1	2481	3707.8	5022.4	6300.3	7630	8586.4	10150
8	1263.9	2409.8	3585.1	4948.5	6274.4	7480	8405.4	9343.2
10	1210.7	2392.1	3459.2	4918.7	6256.2	7290.3	8082.4	9900.3

Table 7
Percentage reduction of $C_{\max}$

$p_r = 0$	← Jobs (n) →							
Mac. (m)	10 (%)	20 (%)	30 (%)	40 (%)	50 (%)	60 (%)	70 (%)	80 (%)
4	19.10	14.83	17.70	15.34	16.08	15.30	17.25	14.84
6	16.04	9.23	10.81	8.02	5.59	6.01	10.87	9.54
8	13.43	11.15	5.26	8.37	5.79	6.66	7.17	10.86
10	15.81	10.68	7.09	7.45	5.76	5.99	7.09	5.34

Table 8
Reducing processing times of Load and Unload operations

$p_r = 0$	← Jobs (n) →							
Mac. (m)	10	20	30	40	50	60	70	80
4	1833.9	3071.5	4312.1	6273.3	7639.3	8816.5	10015.8	11228.9
6	1567.7	2400.4	3607.3	4847.7	5467.2	6682	7717.9	9035.2
8	1334.5	2398.5	3264.2	4475.5	5458.9	6653.9	7527.2	8697.5
10	1301.5	2378.7	3138.8	4298.5	5429.5	6354	7097.6	8415.5

Table 9
Percentage reduction of C_{max}

$p_r = 0$	← Jobs (n) →							
Mac. (m)	10 (%)	20 (%)	30 (%)	40 (%)	50 (%)	60 (%)	70 (%)	80 (%)
4	7.28	5.94	7.38	7.40	5.81	5.20	6.49	7.02
6	6.78	12.18	13.23	11.22	18.07	17.69	19.89	19.48
8	8.60	11.56	13.74	17.13	18.03	16.97	16.87	17.02
10	9.50	11.18	15.70	19.12	18.21	18.06	18.41	19.53

tasks and tasks execution requires simultaneous usage of several resources. The case of an automatic assembly cell is examined. The NP-completeness in the strong sense of the problem is proved for an automatic assembly cell with four machines. A heuristic algorithm is proposed. For this we give computational results for an assembly cell with different number of machines and one robot. The procedure at each iteration selects a task, based on the partial schedule obtained for the parts that had already been loaded for the assembly process. That characteristic of the proposed algorithm indicates that the presented approach can also be applied in on-line scenarios as well as in dynamic scheduling environment.

Further research will be devoted to the extension of this approach to different cell configuration, such as the case in which a task requires k additional resources out of a set of m available ones.

References

[1] Agnetis A, Pacciarelli D. Part sequencing in three-machine no-wait robotic cells. Oper Res Lett 2000;27:185–92.

[2] Aneja YP, Kamoun H. Scheduling of parts and robot activities in a two-machine robotic cell. Comput Oper Res 1999;26(4): 297–312.

[3] Brauner N, Finke G. Optimal moves of the material handling system in a robotic cell. Int J Prod Econom 2001;74:269–77.

[4] Hall NG, Kamoun H, Sriskandarajah C. Scheduling in robotic cells: complexity and steady state analysis. Eur J Oper Res 1998;109:43–65.

[5] Kats V, Levit VE, Levner E. An improved algorithm for cyclic flowshop scheduling in a robotic cell. Eur J Oper Res 1997;97:500–8.

[6] Agnetis A. Scheduling no-wait robotic cells with two and three machines. Eur J Oper Res 2000;123(2):303–14.

[7] Agnetis A, Arbib C, Lucertini M, Nicolò F. Part routing in flexible assembly systems. IEEE Trans Robotics Automation 1990;6(6):697–705.

[8] Agnetis A, Lucertini M, Nicolò F. Flow management in flexible manufacturing cells with pipeline operations. Manage Sci 1993;39(3):294–306.

[9] Agnetis A, Macchiaroli R. Modelling and optimization of the assembly process in a flexible cell for aircraft panel manufacturing. Int J Prod Res 1998;36(3):815–30.

[10] Askin RG, Standridge CR. Modelling and analysis of manufacturing systems. New York: Wiley; 1993.

[11] Van De Klundert J. Scheduling problem in automated manufacturing. Dissertation no. 96–35, Faculty of Economics and Business Administration, University of Limburg, Maastricht, 1996.

[12] Asfahl CR. Robots and manufacturing automation. New York: Wiley; 1985.

[13] Sethi SP, Sriskandarajah C, Sorger G, Blazewicz J, Kubiak W. Sequencing of parts and robot moves in a robotic cell. Int J Flexible Manuf Systems 1992;4:331–58.

[14] Hall NG, Kamoun H, Sriskandarajah C. Scheduling in robotic cells: classification, two and three machine cells. Oper Res 1997;45(3):421–39.

[15] Blazewicz J, Kubiak W, Sethi SP, Sorger G, Srikandarajah C. Sequencing of parts and robot moves in a robotic cell. Int J Flexible Manuf Systems 1992;4:331–58.

[16] Hartley J. Robots at work. Amsterdam: North-Holland; 1983.

[17] Wilhelm WE. Complexity of sequencing tasks in assembly cells attended by one or two robots. Naval Res Logist 1987;34: 3447–63.

[18] Crama Y, Van De Klundert J. Robotic flowshop scheduling is strongly NP-complete. In: Klein Haneveld WK, Vrieze OJ, Kallenberg LCM, editors. Ten Years LNMB. Amsterdam, The Netherlands: CWI Tract 122; 1997. p. 277–86.

[19] Levner E, Kats V, Levit VE. An improved algorithm for cyclic flowshop scheduling in a robotic cell. Eur J Oper Res 1997;97:500–8.

[20] Crama Y, Van De Klundert J. Cyclic scheduling of identical parts in a robotic cell. Oper Res 1997;45:952–65.

[21] Glass CA, Shafransky YM, Strusevich VA. Scheduling for parallel dedicated machines with a single server. Naval Res Logist 2000;47:304–28.

[22] Hall NG, Potts CN, Sriskandarajah C. Parallel machine scheduling with a common server. Discrete Appl Math 2000;102(3): 223–43.

[23] Hertz A, Mottet Y, Rochat Y. On a scheduling problem in a robotized analytical system. Discrete Appl Math 1996;65: 285–318.

[24] Jeng WD, Lin JT, Wen UP. Algorithms for sequencing robot activities in a robot-centred parallel-processor workcell. Comput Oper Res 1993;20(2):185–97.

[25] Baker KR. Introduction to sequencing and scheduling. New York: Wiley; 1976.

[26] Chu C, Proth JM. Single machine scheduling with chain structures procedence constraints and separation time windows. IEEE Trans Robotics Automation 1996;12(6):835–44.

[27] Kise H. On an automated two-machines flowshop scheduling problem with infinite buffer. J Oper Res Soc Jpn 1991;34(3): 354–61.

[28] Kise H, Shioyama T, Ibaraki T. Automated two-machines flowshop scheduling: a solvable case. IIE Trans 1991;23(1): 10–6.

[29] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. San Francisco: W.H. Freeman; 1979.