Scheduling of coupled tasks and one-machine no-wait robotic cells

Nadia Brauner, Gerd Finke, Vassilissa Lehoux-Lebacque[*]
Laboratoire G-SCOP, 46 av. Félix Viallet, 38031 Grenoble, France

Chris Potts, Jonathan Whitehead
School of Mathematics, University of Southampton
Highfield, Southampton SO17 1BJ, UK

Abstract

Coupled task scheduling problems were first studied more than 25 years ago. Several complexity results have been established in the meantime, but the status of the identical task case still remains unsettled. We describe a new class of equivalent one-machine no-wait robotic cell problems. It turns out that scheduling of identical coupled tasks corresponds to the production of a single part type in a robotic cell. We describe new algorithmic procedures to solve this robotic cell problem, allowing lower and upper bounds on the production time, and discussing in particular cyclic production plans.

This research has been supported by INTAS Project 03-51-5501

This paper has appeared as a journal publication in:
Computers & Operations Research, Volume 36, Issue 2, pages 301-307, February 2009.
For information about the journal, see:
http://www.elsevier.com/wps/find/journaldescription.cws_home/300/description#description

[*]vassilissa.lebacque@g-scop.inpg.fr, Tel: (33) 4 76 57 48 42, Fax: (33) 4 76 57 46 02

1 Coupled tasks

A set of n coupled tasks are to be scheduled on a single machine. A coupled task j consists of two operations to be executed on the machine which have processing times a_j and b_j. These two operations have to be executed in a specified order and have a separation time of exactly L_j time units (the time between the completion of the first operation and the start of the second). The objective is to minimize the makespan $C_{\max}$ or, in the case of a cyclic schedule, to maximize the throughput rate. Let us denote this problem as Problem (C).

The coupled-task problem was first studied by Shapiro [7] in the context of scheduling operations for radar, while Orman *et al.* [4] present a more detailed case study involving a multifunction radar system. For radar scheduling applications, the first task is a pulse transmission and the second task is a pulse reception. The separation between tasks is the time for a pulse of energy to be transmitted to a potential target and then be reflected back to the radar. The radar system studied by Orman *et al.* [4] is used in a military environment for weapon guidance, tracking targets, and surveying volumes of space to find targets.

Gupta [2] describes several other potential applications of the model. In a chemical plant where the same processor is used to perform several operations on the same job, a specified amount of time must elapse between operations due to the chemical reactions involved. A further application occurs for the scheduling of a workstation within a flexible manufacturing system. The workstation can perform different types of operations, although refixturing of the part is needed between successive operations. This refixturing is performed at a load/unload station, and while this is undertaken the workstation can be used to process other jobs.

There are various complexity results for this problem (Orman and Potts [5]), but the complexity for identical tasks ($a_j = a$, $L_j = L$, $b_j = b$) is still open (see Table 1 from [5]). The best known algorithm for n identical tasks, to our knowledge, has complexity $O(nr^{2L})$, where $r \leq \sqrt[a-1]{a}$ (Ahr *et al.* [1]). Note, however, that this is not an algorithm for which the time complexity is polynomial in the size of the instance. This would be a polynomial in $O(\log \max\{n, a, L, b\})$, or $O(\log \max\{a, L, b\})$ in the cyclic case.

Table 1 about here

In the subsequent sections of the paper, we describe a robotic cell problem which is equivalent to the coupled-task problem (Sections 2 and 3). In Section 4, we report numerical results for problems where the separation times have a certain tolerance. Section 5 treats the case of a cyclic production.

2 One-machine no-wait robotic cell

We consider a robotic cell composed of an input station (IN), an output station (OUT) and a machine (M). IN and OUT have infinite capacity to store the raw material for the parts to be produced and the finished parts, respectively. The machine M can treat any number of parts simultaneously (for instance in a chemical tank). Note that this assumption is not usually made in the robotic cell literature. The configuration of this type of robotic cell is shown in Figure 1.

Figure 1 about here

A single transporter, a robot, carries out the material handling between IN, M and OUT. It has unit capacity (it can carry one part at a time), but it can bring a part to M, leave it there and pick up a finished part that is transported to OUT.

A task T_j consists of the following operations: the corresponding material is to be transported from IN to M in A_j time units that may in the general case depend on j. The part is then processed for p_j time units at M, and after being processed has to be transferred without delay (no-wait case) in B_j time units to OUT. For the complete process, we also consider the movements of the empty robot from M to IN (α time units), from OUT to M (β time units) and from OUT to IN (γ time units). The objective is to execute n tasks with minimal makespan $C_{\max}$, or in the cyclic case, to maximize the throughput rate (for instance for given proportions of the different part types).

Different classes of this problem may be defined by imposing various conditions on the travel times of the empty robot. We distinguish the following cases:

(R_a) additive problem: $\gamma = \alpha + \beta$,

(R_e) equidistant problem: $\alpha = \beta$ and $\gamma < \alpha + \beta$,

(R_g) general problem with arbitrary α, β, γ.

In particular, it has been noticed by Crama and Dror (private communication) that the complexity status of problems (R_a) and (R_e) is open for identical parts.

3 Two equivalent problems

We call two optimization problems Π_1 and Π_2 equivalent if there are mappings from the set of feasible solutions of Π_1 to the feasible solutions of Π_2 and vice versa, that preserve optimality.

Theorem 1 *The coupled-task problem* (C) *is equivalent to the one-machine no-wait robotic cell problem* (R_a).

Proof. Define the two following activities for the robot:

- U_j: the empty robot goes from M to IN (α time units), picks up, carries and unloads the material corresponding to task T_j at M (A_j time units);
- V_j: the robot loads the part corresponding to task T_j from M and carries it to OUT (B_j time units). Then, the empty robot returns to M (β time units).

Without loss of generality, we suppose that the robot only waits at the machine (if required). Therefore, the preceding activities do not contain any waiting time. While at machine M, the robot can only execute one of the two activities described above. Note that we use the fact that $\gamma = \alpha + \beta$ since in the movement from OUT to IN, we suppose that the robot passes through M. The no-wait condition imposes that the time elapsed between the activities U_j and V_j is exactly p_j. Setting $a_j = \alpha + A_j$, $b_j = \beta + B_j$ and $L_j = p_j$, we get the corresponding instance of (C). The scheduling in (C) is exactly the same as for the part production in (R_a).

Conversely, starting with an instance $\{a_j, L_j, b_j\}$ of (C), we may define return trips of zero duration for the empty robot (in fact, any times α and β such that $0 \leq \alpha \leq \min a_j$ and $0 \leq \beta \leq \min b_j$ can be used). We set $A_j = a_j - \alpha$, $B_j = b_j - \beta$, and $\gamma = \alpha + \beta$. Again, the two problems are the same. $\square$

From the complexity results on the coupled-tasks problem shown in Table 1, we can deduce the complexity results for the robotic cell scheduling problem (R_a) listed in Table 2.

Table 2 about here

4 Coupled tasks with tolerance

We now consider coupled tasks with tolerance, which allow separation times to vary between L_j and $L_j + \delta_j$. This case is particularly important in connection with one-machine no-wait robotic cells since the interval $[L_j, L_j + \delta_j]$ corresponds to a production time with lower and upper bounds. This is the usual situation in problems related to hoist scheduling where the machines correspond to chemical tanks.

This problem is considered by Potts and Whitehead [6]. Since the problem is strongly NP-hard, they develop local search and construction algorithms to tackle it. The local search algorithms (multiple restart descent, iterated descent and tabu search) all use a common solution representation and neighborhood structure. Solutions are represented as permutations of the $2n$ operations, representing the exact sequence in which the operations must be scheduled. Since for any given permutation there may not be a corresponding feasible schedule, its feasibility and associated minimum makespan must be determined. A longest path formulation is developed to evaluate solutions; a single solution can be evaluated in $O(n^2)$ time. Various speed-up techniques are proposed to make evaluation of solutions faster in practice. The neighborhood structure is defined by insert moves which move single operations within the permutation representing the current solution.

The authors also develop an $O(n^2)$ construction algorithm. Pairs of coupled tasks are iteratively interleaved to make new coupled tasks. The interleaving process sometimes requires the inclusion of forced idle time. Since the objective of minimizing makespan is equivalent to minimizing idle time, the amount of forced idle time is a crucial factor in choosing which pairs of coupled tasks to join at each iteration. The algorithm also uses other measures to aid the decision. A deterministic version and a randomized version of the algorithm are developed; at each iteration, the latter probabilistically chooses the rules used to determine which pairs of coupled tasks should be joined.

Extensive computational tests are carried out on random instances ranging from n=20 to 100 coupled tasks (i.e. 40 to 200 operations). The instances are split into two groups according to the flexibility, δ_j, of the coupled tasks. In one group, for any given instance the values of δ_j for all of the n coupled tasks are similar, lying within a small range. In the other group, the values of δ_j for the coupled tasks in any given instance may vary significantly. For each instance, each local search algorithm is run 5 times, each run lasting 5 minutes or until a lower bound on the instance is achieved. The deterministic construction heuristic is run only once, whereas the randomized construction heuristic is repeatedly run until 5 minutes have elapsed.

The results of the computational tests of Potts and Whitehead [6] are summarized in Table 3, where $C_{\max}^{\text{ALG}}$ indicates the makespan obtained by the tested algorithm and $C_{\max}^{\text{LB}}$ is a lower bound. The authors comment that the randomized construction heuristic (RC) is the most effective of the tested algorithms. Moreover, both the randomized (RC) and deterministic versions (C) of the construction algorithms show consistent performance as n increases. In contrast, the local search algorithms perform less well and exhibit deteriorating performance as n increases. The authors suggest that this is because of the high computational cost of searching the neighborhood. The most effective local search algorithm is the multiple restart descent (MRD), followed by tabu search algorithms (TS), with the iterated descent (ID) yielding the poorest performance.

Table 3 about here

5 Cyclic production

Figure 2 about here

We now return to the main case of Problem (C) for identical parts (a, L, b), where a, L and b are integers. We may assume that $a \geq b$, otherwise we consider the reverse problem. For the equivalent one-machine no-wait robotic cell, a single part type is produced. In this case, it is quite common to try to channel as fast as possible the maximum number of parts through the cell. The number of parts n is very large or not determined in advance. The objective is to determine a production pattern, which can be repeated identically (i.e. find a *production cycle*) that maximizes the throughput rate of the part.

We want to adapt the algorithm of Ahr *et al.* [1] so that it solves this cyclic case. We briefly outline the method in order to explain the necessary modifications. For a given instance (a, L, b) and a given sequence of the tasks, one may associate a *pattern* of length L with each task as shown in Figure 2). A '0' indicates that this position of L is not occupied, and a block in which '1' is repeated b times in succession means that this portion of L is occupied by the b-part of a predecessor task in the sequence. Since each two-operation coupled task is completely rigid, knowledge of the position of the b-part fixes the previous position of the a-part at distance L. In Figure 2 we have for the two middle tasks the patterns

$$p_1 = 00000110 \qquad p_2 = 11011000.$$

The pattern p_2 belongs to the immediate successor of p_1. Therefore, one knows that the last block of '1' in p_2 must necessarily indicate the position of the b-part of the predecessor task p_1. Consequently, one also has the length of the right shift l between p_1 and p_2. In this way, a given sequence of patterns uniquely defines the schedule of the parts and, conversely, each schedule gives the pattern sequence.

The following directed graph $G = (V, E)$ is introduced. The vertex set V consists of all possible patterns and there is an arc from pattern p_i to p_j with weight l if the placement of the corresponding tasks yields a right-shift of l units (as explained for p_1 and p_2 in Figure 2). In Ahr *et al.* [1] only an approximate value for the number of vertices $|V|$ is given. The exact number is obtained by the following formula:

$$|V| = 1 + \sum_{i=1}^{\lfloor \frac{L}{a} \rfloor} \prod_{k=1}^{i} \frac{L - ia + k}{k}.$$

Finding the optimal placement of n tasks (a, L, b) corresponds to finding the shortest path of length $\mathcal{L}(n)$ of n vertices in graph G, starting at vertex $p_0 = 0$. Then the optimal schedule length is given by $\mathcal{L}(n) + (a + L + b)$. In Ahr *et al.* [1], determining the shortest path is done in a rather uneconomical manner, where for each vertex all successors together with the different arc weights are memorized.

Let us now turn to the problem of finding the optimal production cycle. As an illustration, consider the instance $a = 3$, $L = 8$ and $b = 2$. In Figures 3–5, three different feasible production cycles are displayed. In Figuer 3, one has the rather obvious placement of the form *aaabbb* with the mean cycle length $L_1 = \frac{19}{3} = 6.\dot{3}$. In Figure 4 the cycle *abab* has mean cycle length $L_2 = \frac{13}{2} = 6.5$. Finally, the optimal cycle *aababbab* is displayed in Figure 5 and has length $L_3 = 21/4 = 5.25$.

Figure 3 about here

Figure 4 about here

Figure 5 about here

A production cycle corresponds to a circuit in graph G. Its mean length $\mathcal{L}_\mu$ is defined as the sum of its arc weights divided by the number of vertices. Our cyclic production problem is solved by a circuit with minimal $\mathcal{L}_\mu$. This combinatorial problem is well known and solved efficiently in the literature and goes under the name of the *shortest mean cycle problem* (Karp [3]).

Concerning an appropriate data structure for the implementation of the algorithm, one can make the following observation. Let a pattern p_2 be given. Then the last block of '1' in p_2 belongs to the b-part of **every predecessor vertex** p_1. Therefore, all arcs arriving at p_2 have an identical arc weight l. Note that for the case $p_2 = 0$, a new task is placed without nesting so that $l = a + L + b$. It is much more efficient to simply put this common arc weight on the vertex and to work with vertex weights instead of arc weights. With this structure, one can eliminate a large number of vertices and arcs. In fact, it is sufficient to consider only vertices with weight $l = a$ or $l \geq a + b$. Otherwise, for $a < l < a + b$ and $b \geq 2$, one cannot place any other task between the a-parts and b-parts (see Figure 6). Therefore, one gets a better schedule by shifting vertex i to the left to $l = a$. Thus, we can eliminate all vertices with weights $l \in \{a + 1, a + 2, \ldots, a + b - 1\}$.

Figure 6 about here

A glance at the number of vertices shows that the degree of difficulty of the problem is influenced by the size of $\left\lfloor \frac{L}{a} \right\rfloor$. The size of the graph explodes with increasing values $\left\lfloor \frac{L}{a} \right\rfloor$. Table 4 gives an indication of the solvability of the cyclic production problem (on a Pentium 4, CPU 2.53GHz, RAM 1Gb).

Table 4 about here

6 Conclusion

We have established a new equivalence between the coupled task problem and a certain type of 1-machine robotic cell problem. The known complexity results carry over to these robotic cell problems. It still remains the challenge to settle the status of the identical part problem (a, L, b), which is open for more than twenty years. We report new algorithmic procedures for this problem, with and without tolerances on the distance L.

References

[1] Ahr D, Békési J, Galambos G, Oswald M, Reinelt G. An exact algorithm for scheduling identical coupled tasks. Mathematical Methods of Operations Research (ZOR) 2004; 59: 193–203.

[2] Gupta JND. Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags. Journal of Global Optimization 1996; 9: 239–250.

[3] Karp M. A characterization of the minimum cycle mean in a digraph. Discrete Mathematics 1978; 23: 309–311.

[4] Orman AJ, Potts CN, Shahani AK, Moore AR. Scheduling for a multifunction phased array radar system. European Journal of Operational Research 1996; 90: 13–25.

[5] Orman AJ, Potts CN. On the Complexity of Coupled-task Scheduling. Discrete Applied Mathematics 1997; 72: 141–154.

[6] Potts CN, Whitehead JD. Heuristics for a coupled-operation scheduling problem. Journal of the Operational Research Society *In press*; doi: 10.1057/palgrave.jors.2602272.

[7] Shapiro RD. Scheduling coupled tasks. Naval Research Logistics Quarterly 1980; 27(2): 489–497.

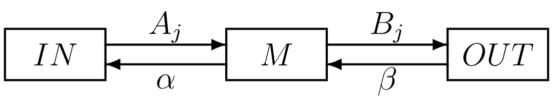

Figure 1: 1-machine robotic cell

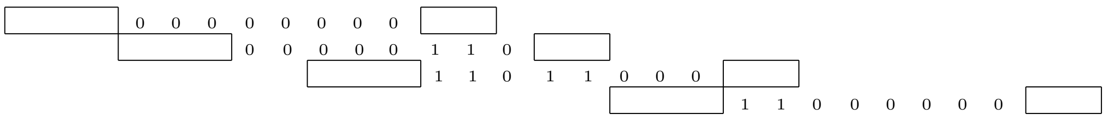

Figure 2: Sequences of patterns

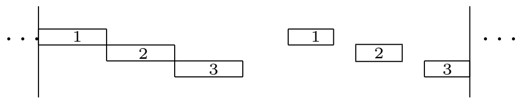

Figure 3: $L = 8$, $a = 3$, $b = 2$, solution $\ldots aaabbb \ldots$, 3-part cycle, mean cycle time $L_1 = 19/3$

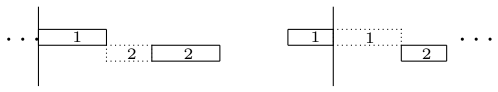

Figure 4: $L = 8$, $a = 3$, $b = 2$, solution $\ldots abab \ldots$, 2-part cycle, mean cycle time $L_2 = 13/2$

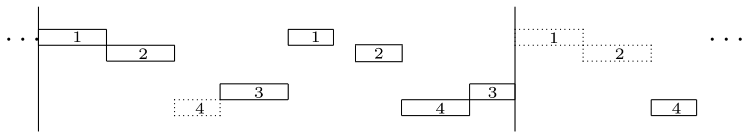

Figure 5: $L = 8$, $a = 3$, $b = 2$, solution $\ldots aababbab \ldots$, 4-part cycle, mean cycle time $L_3 = 21/4$

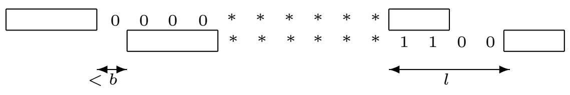

Figure 6: Graph reduction for $a < l < a + b$

Table 1: Table of complexities

Strongly NP-hard	a_j; L_j; b_j
	$a_j = L_j = b_j$
	$a_j = a$; L_j; $b_j = b$
	$a_j = a$; $L_j = L$; b_j
Open	$a_j = a$; $L_j = L$; $b_j = b$
Polynomial	$a_j = L_j = p$; b_j
	$a_j = b_j = p$; $L_j = L$

Table 2: Table of complexities of problem (R_a)

Strongly NP-hard	A_j; p_j; B_j	multiple parts, part dependent travel
	$A_j+\alpha=p_j=B_j+\beta$	related travel & processing, multiple parts
	$A_j=A$; p_j; $B_j=B$	multiple parts, constant travel
	$A_j=A$; $p_j=p$; B_j	identical processing
Open	$A_j=A$; $p_j=p$; $B_j=B$	identical parts
Polynomial	$A_j+\alpha=p_j=p$; B_j	identical processing,
	$A_j+\alpha=B_j+\beta=p_j=p$	related travel & processing

Table 3: Average algorithm performance ratio, $C_{\max}^{\mathrm{ALG}}/C_{\max}^{\mathrm{LB}}$, of heuristics for the coupled-task scheduling problem with tolerances

	$n = 20$	$n = 30$	$n = 50$	$n = 100$	Overall
RC	1.046	1.045	1.043	1.040	1.043
C	1.057	1.058	1.054	1.044	1.053
MRD	1.047	1.058	1.068	1.115	1.072
TS	1.051	1.063	1.072	1.113	1.075
ID	1.077	1.091	1.091	1.126	1.096

Table 4: Computational results with $a = 5, b = 3$ and varying L

L	$\lfloor L/a \rfloor$	number of vertices	reduced number of vertices	solution time
10	2	8	6	$<$ 1 sec
12	2	15	8	$<$ 1 sec
14	2	26	14	$<$ 1 sec
16	3	45	22	$<$ 1 sec
18	3	80	36	$<$ 1 sec
20	4	140	58	$<$ 1 sec
22	4	245	92	$<$ 1 sec
24	4	431	151	$<$ 1 sec
26	5	756	241	$<$ 1 sec
28	5	1326	391	$<$ 1 sec
30	6	2328	627	$<$ 1 sec
32	6	4085	1013	$<$ 1 sec
33	6	5441	1288	$<$ 1 min
34	6	7168	1634	$<$ 1 min
35	7	9496	2071	$<$ 1 min
36	7	12580	2632	$<$ 1 min
37	7	16665	3344	$<$ 1 min
38	7	22076	4246	$<$ 1 min
39	7	29244	5389	$<$ 1 hour
40	8	38740	6839	$<$ 1 hour
41	8	51320	8688	$<$ 1 hour
42	8	67985	11034	$<$ 1 hour
43	8	90061	14007	not solved