Math Meth Oper Res (2004) 59:193–203
DOI 10.1007/s001860300328

Mathematical Methods
of Operations Research

An exact algorithm for scheduling identical coupled tasks

Dino Ahr[1], József Békési[2], Gábor Galambos[2], Marcus Oswald[1], Gerhard Reinelt[1]*

[1] Institute of Computer Science, University of Heidelberg, Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany
[2] Department of Informatics, Juhász Gyula Teacher's Training College, University of Szeged, Pf. 396, H-6720 Szeged, Hungary

Manuscript received: April 2003/Final version received: October 2003

Abstract. The coupled task problem is to schedule n jobs on one machine where each job consists of two subtasks with required delay time between them. The objective is to minimize the makespan. This problem was analyzed in depth by Orman and Potts [3]. They investigated the complexity of different cases depending on the lengths a_i and b_i of the two subtasks and the delay time L_i. $\mathcal{NP}$-hardness proofs or polynomial algorithms were given for all cases except for the one where $a_i = a$, $b_i = b$ and $L_i = L$. In this paper we present an exact algorithm for this problem with time complexity $O(nr^{2L})$ where $r \leq \sqrt[a-1]{a}$ holds. Therefore the algorithm is linear in the number of jobs for fixed L.

Key words: Scheduling, Coupled Tasks, Dynamic Programming

1 Introduction

The coupled task problem can be defined as follows. We are given n jobs each of them consisting of two distinct tasks (operations). The sequence of these tasks is fixed and also a fixed delay time has to pass between the two tasks. So, each job i can be denoted by a triple (a_i, L_i, b_i), where the values represent the processing time of the first task, the delay time between the tasks and the processing time of the second task, respectively. It is important that in this type of problem the second task must be scheduled *exactly* $L_i + a_i$ units after the start of the first one. During the delay time the machine is idle and so it can process other jobs in this interval. The aim is to schedule n coupled tasks on one machine in such a way that no two tasks overlap and the latest finishing time of the jobs is minimized. (This time is called makespan and

* Research was supported by DAAD exchange program 324 PPP-Ungarn.

denoted by $C_{\max}$). Preemption is not allowed, i.e., every subtask has to be processed continuously.

Several applications of the problem are discussed in [3].

Scheduling problems are normally classified using the standard three field notation $\alpha|\beta|\gamma$, introduced by Graham et al. in [1]. Here the first field describes the machine environment, the second the job characteristics and the last one is the optimality criterion. Following the idea in [3] we write in the second field *Coup-Task* indicating that all jobs in fact consist of coupled tasks. With this formulation (1|Coup-Task, $a_i = a, L_i = L, b_i = b|C_{\max}$) denotes a coupled task problem where all jobs are identical, the so-called *Identical Coupled Task Problem (ICTP)*.

The general problem (1|Coup-Task|$C_{\max}$) is $\mathcal{NP}$-hard [4]. Shapiro [5] discussed practical situations where the problem arises and gave three simple heuristics. Unfortunately, these heuristics have not been analyzed in detail. Only experimental results showed their efficiency. Orman and Potts [3] studied the problem from a complexity point of view. They covered all subcases except for the identical case $a_i = a$, $L_i = L$, $b_i = b$. Fig. 1 presents an example of an optimal schedule for the identical case.

In this paper we investigate this special case. Section 2 contains some basic definitions and lemmas for helping to understand the idea of our solution. Section 3 describes a graph-theoretic model which leads to a solution algorithm for the problem based on shortest path computations. Implementation issues and computational results are discussed in Sect. 4. In Sect. 5 the structure of optimal schedules is analysed. The formulation of some open problems concludes the paper.

2 Preliminaries

Throughout the paper we denote by a and b the operation time of the first and the second task of a job, by L the common delay time, and by n the number of jobs. We always assume that a, b and L are integers, because we can convert rationals to integers by multiplying with a suitable number.

Without losing generality we can assume that $a \geq b$, since the following theorem (cf. [3]) holds.

Theorem 1 *For given a and b the problem* (1|Coup-Task, $a_i = a, L_i = L, b_i = b|C_{\max}$) *and its reverse* (1|Coup-Task, $a_i = b, L_i = L, b_i = a|C_{\max}$) *are equivalent.*

We may also assume that $a < L < (n-1)a$, since otherwise the simple greedy algorithm (i.e., start the first job at time 0, and each subsequent job as soon as possible) already gives the optimal solution.

We will use certain patterns to represent idle and busy times of the processor with respect to a given schedule. These patterns consist of 0's and 1's indicating if the processor is idle or busy during a certain time unit.

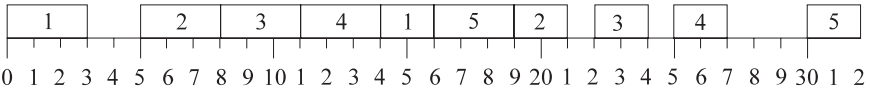

Fig. 1. An optimal schedule for 5 jobs with values $a = 3$, $b = 2$, $L = 11$

Definition 1 *A* 0-1 *sequence of length L is called* $P(a,b,L)$ *pattern, if it contains* 1*'s only in blocks of length b and if each such block is followed by at least* $a-b$ 0*'s.*

In our algorithm we will schedule the jobs one after the other and will use $P(a,b,L)$ patterns to describe the processor status in the gap of the job scheduled last so far.

For example, the sequence 11000110110 is a $P(3,2,11)$ pattern, but the sequence 11011100110 is not, because it contains a block of 1's of length 3. The set of all possible $P(3,2,7)$ patterns is $\{0000000, 1100000, 0110000, 0011000, k\ 0001100, 0000110, 1101100, 1100110, 0110110\}$.

Lemma 1 *The total number* $np(a,b,L)$ *of possible* $P(a,b,L)$ *patterns is* $O(r^L)$*, where r is the absolute value of the root with the greatest absolute value of the equation* $1+x^{a-1}=x^a$*. For* $a \geq 2$ *an upper bound of r is given by* $\sqrt[a-1]{a}$.

Proof. If $L < a$ we have $np(a,b,L)=1$, since in this case only the pattern $0\ldots0$ satisfies the condition.

We show the recursion $np(a,b,L)=np(a,b,L-1)+np(a,b,L-a)$ for $L \geq a$. In this case the patterns can be classified into two groups. The first group contains those patterns, which end with a block of 1's and $a-b$ 0's. The second group consists of patterns which end with more than $a-b$ 0's. The number of patterns in the first group obviously is $np(a,b,L-a)$ and the second group contains $np(a,b,L-1)$ elements. And therefore the recursion holds.

The characteristic equation of the recursion is $1+x^{a-1}=x^a$ and the first part of the lemma follows by the well-known solution method of linear recursions.

It remains to show that $\sqrt[a-1]{a}$ is an upper bound for r if $a \geq 2$. First of all we show that $\sqrt[a-1]{a} > 1+\frac{1}{a}$ for all $a \geq 2$. For $a=2$ this follows from a simple calculation, for $a \geq 3$ it follows from the well-known fact that $(1+\frac{1}{a})^{a-1} < e < a$. Now we assume that $r > \sqrt[a-1]{a}$. Then we have

$$r^a - r^{a-1} = r^{a-1}(r-1) > a(\sqrt[a-1]{a}-1) > a(1+\frac{1}{a}-1) = 1.$$

On the other hand with r being the greatest absolute value of a root of $1+x^{a-1}=x^a$ the triangular inequality yields $1+r^{a-1} \geq r^a$ which is a contradiction. □

Suppose we have started exactly k jobs and the schedule has the property, that no new job can be started before the first task of the last job. This means that job $k+1$ can only be started after the start of the last job, possibly before but in any case after its second task. The starting time of the new job depends on the idle time periods between the two tasks of the last job. We can represent the schedule in the gap of the last job by a $P(a,b,L)$ pattern. In principle, situations corresponding to every $P(a,b,L)$ pattern are possible. Fig. 2 visualizes the concept of $P(a,b,L)$ patterns.

We now investigate how a further job affects the schedule. It is obvious, that we can start a new job only at those positions, where we have an idle period of length at least a. This means that the corresponding $P(a,b,L)$ pattern contains a block of 0's of length a at that position. If we start the new

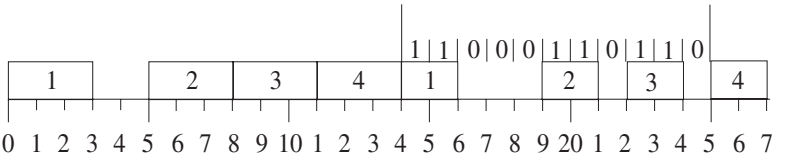

Fig. 2. Gap of last job represented by a $P(3,2,11)$ pattern

job at a given position, then the new gap pattern for the last job (which will be defined as $S(p,i)$ below) will be generated by a rule. First we copy some bits – their number depending on the position of the new job – from the right hand side of the original pattern, then add b bits of 1's and fill the pattern with 0's up to length L. It is also obvious, that the increase of the makespan depends on the starting position of the new job. Fig. 3 illustrates the method of generating the new pattern.

More exactly, we can define the following operator.

Definition 2 *Let p be a $P(a,b,L)$ pattern and i, $1 \leq i \leq L-a+1$, be an integer such that*

$$p[i] = p[i+1] = \ldots = p[i+a-1] = 0. \tag{1}$$

Then $S(p,i)$ is the 0–1 *sequence*

$$p[i+a]\ p[i+a+1]\ \ldots\ p[L]\ 1^b\ 0^{i+a-b-1}$$

where 1^k (0^k) denotes a string of k 1's (0's). Furthermore we define the number $w(S(p,i)) = i+a-1$.

Lemma 2 *If p is a $P(a,b,L)$ pattern, then $S(p,i)$ is a $P(a,b,L)$ pattern for $1 \leq i \leq L-a+1$ and $w(S(p,i))$ gives the increase of the makespan.*

Proof. The statement of the lemma follows directly from the definition of $S(p,i)$, because we replace some right hand side bits of p by one block of 1's of length b followed by at least $a-b$ 0's. Obviously the new last job starts $i+a-1$ time units after the previous last job. □

3 Graph model and algorithm

In the above section we have shown that we can associate with every schedule a $P(a,b,L)$ pattern characterizing the state of the machine in the gap of the job currently started as the last job. It is obvious, that this pattern will usually change, when we add a new job to the schedule.

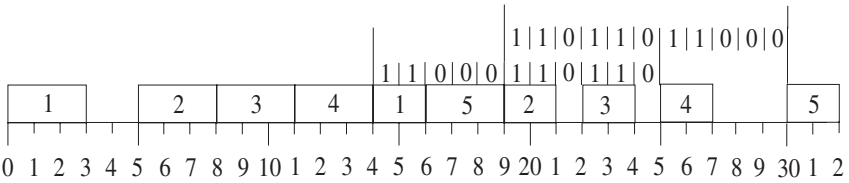

Fig. 3. Generating the new pattern after starting a new job

A natural idea is to represent the relations between the patterns by a directed graph. The vertices of the graph are the possible patterns and we connect two vertices by an arc, if the corresponding pattern can be followed by the other one when a job is added. We can also assign a weight to each arc, giving the increase of the makespan caused by this change of schedule.

Definition 3 *For given integers a, b and L we define the directed graph $G = (V, A)$ with arc weights $w(p, q)$, for every $(p, q) \in A$ as follows.*

– $V := \{p \mid p \text{ is a } P(a, b, L) \text{ pattern}\}$,
– $A := \{(p, q) \mid q = S(p, i) \text{ for } i,\ 1 \leq i \leq L - a + 1\} \cup \{(p, 0^L) \mid p \in V\}$,
– $w(p, q) := \min_{1 \leq i \leq L-a+1}\{w(S(p, i)) \mid S(p, i) = q\}$,
– $w(p, 0^L) := a + b + L$.

Note that edges $(p, 0^L)$ model the possibility to start the next job right after the termination of the second subtask of the last job. In this case the makespan is increased by $a + b + L$.

Fig. 4 shows an example graph. Note that the same pattern may occur repeatedly in a schedule and can even be followed by itself.

Now, the coupled-task problem amounts to finding a minimal weight path in G starting from pattern 0^L consisting of exactly $n - 1$ arcs. Since the weight of each arc represents the increase of the makespan, the total weight of such a path is the real makespan minus the time of the first job, which is $a + b + L$. If we minimize the weights of the paths consisting of $n - 1$ arcs, then we minimize the makespan for n jobs.

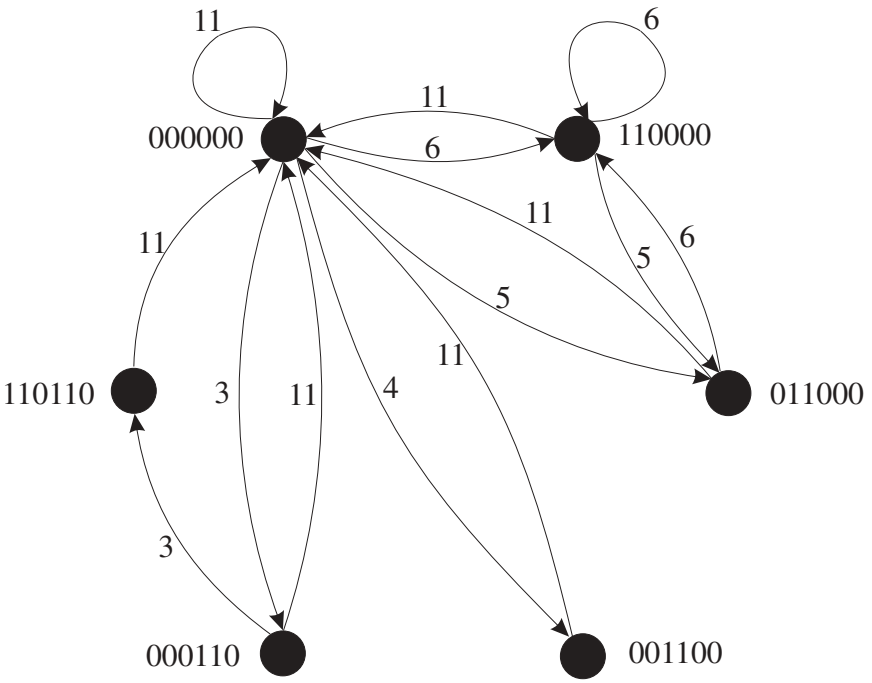

Fig. 4. The weighted pattern graph for $P(3, 2, 6)$ patterns

More precisely, the makespan M of the identical coupled task scheduling problem with parameters a, b, L and n can be expressed as

$$\min_{p \in P(a,b,L)\ patterns} \{w(p_0, p_1, \ldots, p_{n-1}) \mid p_0 = 0^L, (p_i, p_{i+1}) \in A, p_{n-1} = p\}$$

where $w(p_0, p_1, \ldots, p_{n-1}) = \sum_{i=0}^{n-2} w(p_i, p_{i+1})$ and $p_0, \ldots, p_{n-1}$ is a minimal weight path between 0^L and p.

This way our problem reduces to a special shortest path problem in a directed graph with positive arc weights. In the following we deal with this task.

Let $w_{0k}(p)$ denote the shortest path length from 0^L to p with k arcs. The following recursive formula applies for determining $w_{0k}(p)$:

$$w_{0k}(p) = \min_q \{w_{0,k-1}(q) + w(q,p) \mid (q,p) \in A\}.$$

Based on this observation we can give an algorithm computing $w_{0,n-1}(p)$ for each $P(a,b,L)$ pattern. $E(u)$ denotes the set of vertices that are connected to u by an outgoing arc.

Algorithm MinWeight(G, n)

For each $v \in V$ **do**
  For $i := 0$ **to** $n-1$ **do**
    $D_v[i] := \infty$
    $P_v[i] := -1$
$D_{v_0}[0] = 0$ (where v_0 represents the pattern 0^L)
For $i := 1$ **to** $n-1$ **do**
  For each $u \in V$ **do**
    For each $v \in E(u)$ **do**
      if $D_u[i-1] + w(u,v) < D_v[i]$ **then**
        $D_v[i] := D_u[i-1] + w(u,v)$
        $P_v[i] := u$

After executing *MinWeight* we have $w_{0,n-1}(p) = D_p[n-1]$ for each pattern. Let us denote p^* the pattern which corresponds to the endnode of a shortest path having $n-1$ edges. We obtain the length of this path as $D_{p^*}[n-1] = \min\{D_v[n-1] \mid v \in V\}$ and the associated schedule by recursively visiting the nodes $P_v[n-1]$, starting from $v = p^*$. Fig. 5 and Fig. 6 show shortest paths for the example graph depicted in Fig. 4.

Obviously, the time complexity of this algorithm is dominated by time complexity $O(V^2 n)$ of *MinWeight*. Since $|V| = O(r^L)$, the overall complexity of our algorithm is $O(nr^{2L})$ where $r \leq \sqrt[a-1]{a}$. Note that $\sqrt[a-1]{a}$ goes to 1 for increasing a.

4 Implementation techniques and experiments

An important question for the implementation of our algorithm is the efficient generation of the pattern graph. We have to generate all $P(a,b,L)$

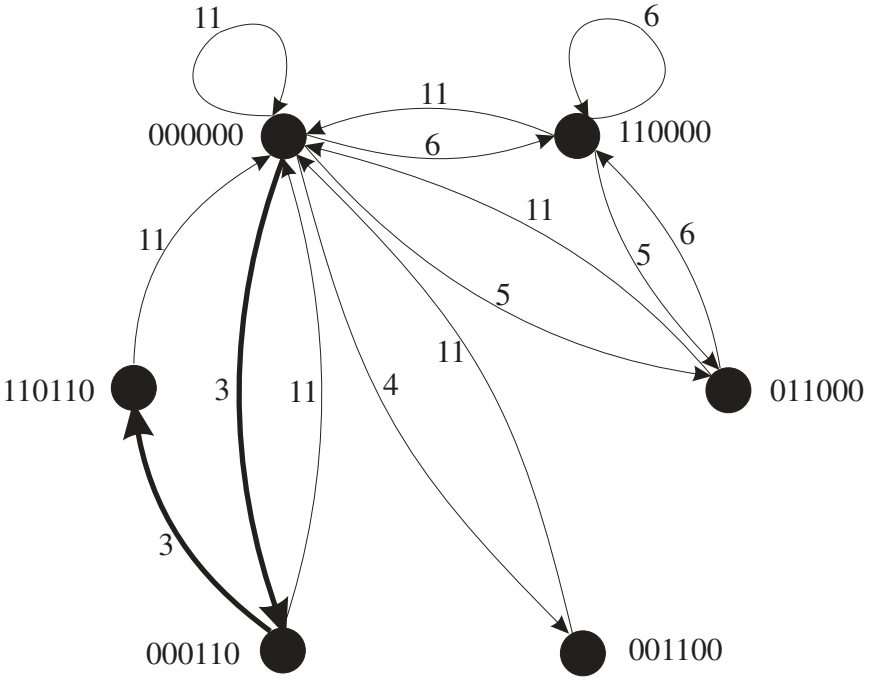

Fig. 5. Shortest path with 2 arcs for the $P(3,2,6)$ pattern graph

patterns and connect them based on operator S, which also has to be realized efficiently. The patterns can be handled as integer numbers where we need a binary representation of at least L bits. If L is small, then we can use built-in arithmetic, otherwise some long arithmetic has to be implemented. Using this binary number representation, the operator S can be defined as follows. Suppose p represents a $P(a,b,L)$ pattern and i, $1 \le i \le L-a+1$, is an integer. Condition (1) can be checked by the relational operator

$$p \bmod 2^{L-i+1} < 2^{L-i-a+1}.$$

If this condition holds, then the formula for obtaining a new pattern is

$$S(p,i) = \left(p \bmod 2^{L-i-a+1}\right)2^{i+a-1} + \left(2^b - 1\right)2^{i+a-b-1}.$$

The following observation helps to decrease the number of necessary edges in the pattern graph.

Lemma 3 *Suppose we have an optimal schedule. Then there exists a schedule with the same makespan which contains no idle period of length larger than* $a+b-2$ *before processing the first subtask of the last job.*

Proof. Let S be an optimal schedule containing a gap longer than $a+b-2$ and let this gap start at time t before the first subtask of the last job is processed.

In the time interval $[t+L+a, t+L+a+b-1]$ the machine either is idle or is occupied by one first subtask of some job or by two first subtasks.

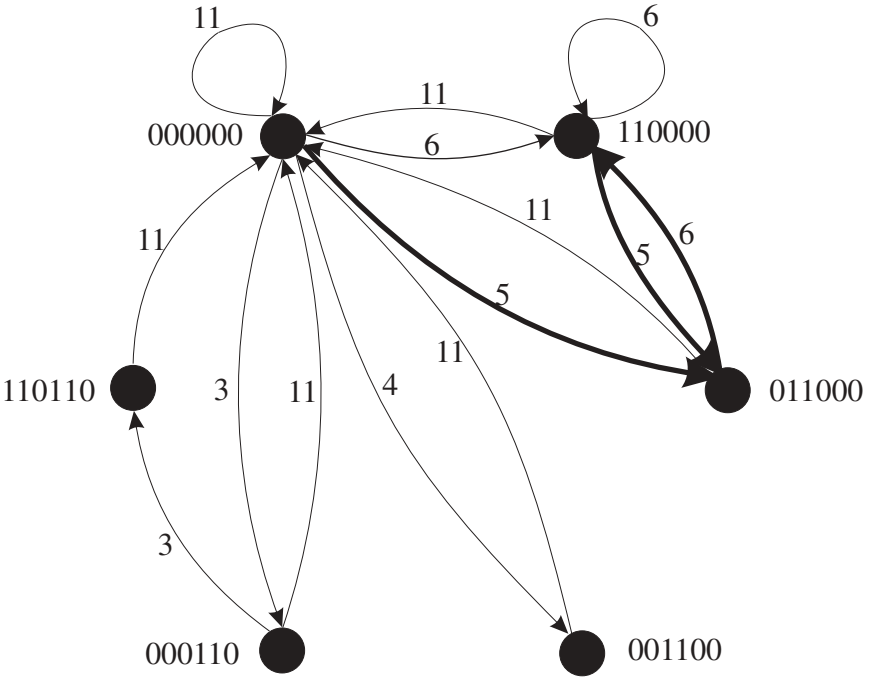

Fig. 6. Shortest path with 3 arcs for the $P(3,2,6)$ pattern graph

In the first case some job can be started earlier than before, in the second case the job being processed in this time interval could have been started at time t. In the third case let $\bar{t}$ be the starting time of the second first subtask. This starting time lies inside the above interval, therefore $t+L+a \leq \bar{t} \leq t+L+a+b-1$ holds. Since the length of the gap starting at time t is at least $a+b-1$, it would have been possible to start the job corresponding to the second first subtask $L+a$ timesteps earlier at a time between t and $t+b-1$. Neither of the three modifications increases the makespan, but eliminates the large gap at time t.

By repeating this method we get an optimal schedule which contains only idle time periods of length at most $a+b-2$, except for gaps occurring after the last job has been started. □

The following algorithm generates the pattern graph $G(V,A)$. It makes use of Lemma 3 to avoid the introduction of unnecessary graph edges.

Algorithm GenGraph(a, b, L)

$V := \{0\}$

$A(0) := \{0\}$

$w(0,0) := a+b+L$

For each unmarked element $u \in V$ **do**

Let J be the indices of bits 1 in $u \cup \{0\}$

For each $j \in J$ **do**
  For $i := j+1$ **to** $j+a+b-1$ **do**
    if $p \bmod 2^{L-i+1} < 2^{L-i-a+1}$ **then**
      $v := (u \bmod 2^{L-i-a+1})2^{i+a-1} + (2^b - 1)2^{i+a-b-1}$
      if v not in V **then**
        Add v to V
      if v not in $A(u)$ **then**
        Add v to $A(u)$
        $w(u,v) := i+a-1$
  Mark u
Output $G(V,A)$

We have implemented our algorithm using the above techniques and tested it for up to 2000 jobs and gap values up to $L = 30$ (in order to be able to use the machine arithmetic). Table 1 displays running times and optimal values on a 433 MHz P-II PC for example problems with $n = 1000$ and $n = 2000$ and also gives optimum values for $n = 1000$. The times show that solutions for fairly large problems can be computed in short time.

5 Structure of optimal schedules

If, for fixed a, b and L, the number of jobs goes to infinity, then the shortest paths in the pattern graph must contain cycles. So, for large n, best paths consist of some first path segment followed by a certain number of repetitions of cycles followed by a terminating segment to meet the number of jobs. Obviously, it makes no sense to repeat cycles other than minimum weight mean cycles, i.e., directed cycles C such that

$$\frac{w(C)}{|C|} = \min\left\{\frac{w(F)}{|F|} \mid F \text{ cycle in } G\right\}.$$

Table 1. Running times of ICTP on a 433 MHz P-II PC

a	b	l	n	Time (ms)	n	Time (ms)	Opt $n = 1000$
4	2	10	1000	60	2000	110	6478
4	2	15	1000	381	2000	751	6971
4	2	20	1000	3074	2000	6149	6460
6	3	20	1000	401	2000	811	9628
6	3	25	1000	2013	2000	4006	9723
6	3	30	1000	12660	1500	18667	9690
8	4	20	1000	140	2000	261	12956
8	4	25	1000	470	2000	941	12284
8	4	30	1000	1770	2000	3566	13942
10	5	20	1000	60	2000	131	17445
10	5	25	1000	190	2000	370	16195
10	5	30	1000	561	2000	1091	14940

Minimum mean cycles in a graph $G = (V, A)$ can be computed in time $O(V(V + A))$ ([2]). Obviously, there may be several minimum weight mean cycles.

So let, for given a, b and L, C be a minimum weight mean cycle with length $l_{a,b,L}$, weight $w_{a,b,L}$ and optimal ratio $r_{a,b,L} = w_{a,b,L}/l_{a,b,L}$. Then we can conclude for the makespan $M(n)$ of an optimal schedule for n jobs that

$$\lim_{n\to\infty} \frac{M(n)}{n} = r_{a,b,L}.$$

Table 2 gives optimal ratios and corresponding minimum weight mean cycle lengths and weights for some problems.

6 Conclusions

In this paper we presented an exact algorithm for the scheduling problem $(1|\text{Coup-Task}, a_i = a, L_i = L, b_i = b|C_{\max})$. Some open questions remain for future research.

A main one is the question of the true complexity of the above problem. Our algorithm is polynomial only in n, so it does not prove that the problem belongs to P.

We conjecture that the problem is polynomially solvable and that it might even be possible to derive an explicit formula for the optimum schedule length depending on a, b and L. So far, however, we did not succeed in proving this.

A first step could be to find a formula for the best weight-length ratio of cycles in the pattern graph. E.g., for $b = 1$, we conjecture that the minimum mean cycle ratio is

$$r_{a,1,L} = \begin{cases} \frac{a+1+2L}{2\frac{L+1}{a+1}} & , \text{ if } \quad L \equiv -1a + 1, \\ \frac{a+1+L}{\lfloor \frac{L}{a+1} \rfloor + 1} & , \text{ if } \quad L \not\equiv -1a + 1. \end{cases}$$

Table 2. Minimum weight mean cycle ratios

a	b	L	$l_{a,b,L}$	$w_{a,b,L}$	$r_{a,b,L}$
3	1	10	3	14	4.666
3	1	15	8	34	4.250
3	1	20	6	24	4.000
4	2	10	4	26	6.500
4	2	15	3	21	7.000
4	2	20	4	26	6.500
6	3	20	3	29	9.666
6	3	25	18	176	9.777
6	3	30	4	39	9.750
8	4	20	4	52	13.000
8	4	25	3	37	12.333
8	4	30	3	42	14.000
10	5	20	2	35	17.500
10	5	25	4	65	16.250
10	5	30	1	15	15.000

Another question is whether the size of the pattern graph can be decreased by eliminating transitions which can only lead to suboptimal schedule.

Furthermore, it would be interesting to construct good heuristics for non-polynomial cases of the coupled-task problem with a thorough worst-case or average-case analysis. There are only very few results published so far in this respect.

Acknowledgements. The authors are grateful to Hans Kellerer who called their attention to this problem.

References

1. Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Annals of Discrete Mathematics 5:287–326
2. Karp RM (1978) A Characterization of the Minimum Cycle Mean in a Digraph. Discrete Mathematics 23:309–311
3. Orman AJ, Potts CN (1997) On the Complexity of Coupled-Task Scheduling. Discrete Applied Mathematics 72:141–154
4. Rinnooy Kan AHG (1976) Machine Scheduling Problems, Martinus Nijhoff, The Hague
5. Shapiro RD (1980) Scheduling Coupled Tasks. Naval Research Logistics Quarterly 20: 489–498