Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Scheduling Coupled-Tasks on a Single Machine

Haibing Li and Hairong Zhao

***Abstract*—In this paper, we consider the coupled-task scheduling problem, to schedule *n* jobs on a single machine. Each job consists of two coupled tasks which have to be processed in a predetermined order and at exactly a specified interval apart. The objective is to minimize the makespan. The problem was shown to be NP-hard in the strong sense even for some special cases. We analyze some heuristics with worst-case bounds for some NP-hard cases. In addition, we present a tabu search meta-heuristic for solving the general case. Computational results show that the meta-heuristic is efficient to solve the problem in terms of solution quality and running time.**

I. Introduction

We consider the coupled-task scheduling problem which is stated as follows. There are n jobs to be scheduled on a single machine. Each job consists of two distinct tasks (operations) which have to be processed in a predetermined order and at a specified interval (or delay) apart. For convenience, each job *j* is denoted by a triple (a_j, L_j, b_j), which represents the first task (also its processing time), the fixed interval (also its length) between the two tasks, and the second task (also its processing time), respectively. It is important that for each job *j*, task b_j must be scheduled *exactly* L_j after the completion of task a_j. Without loss of generality, we assume that during the interval L_j between a_j and b_j, the machine can process the tasks of other jobs. However, at any time the machine can process at most one task and no preemption is allowed. We are interested in minimizing the total schedule length of the jobs, i.e., the makespan C_{max}. Following the same notation in [7], we denote the problem as $1 \mid \text{Coup-Task} \mid C_{\max}$.

The problem arises from some practical applications such as radar system [7, 10]. Unfortunately, this problem is known to be NP-hard [9]. In fact, Orman and Potts [7] showed that even the special cases of this problem, i.e., $1 \mid \text{Coup-Task}, a_j = b_j = L_j \mid C_{\max}$, $1 \mid \text{Coup-Task}, a_j = a, L_j = l \mid C_{\max}$, $1 \mid \text{Coup-Task}, b_j = b, L_j = l \mid C_{\max}$, $1 \mid \text{Coup-Task}, a_j = a, b_j = b \mid C_{\max}$, are all strongly NP-hard. Finally, the very restricted case $1 \mid \text{Coup} - \text{Task}, a_j = b_j = 1 \mid C_{\max}$ has also shown to be strongly NP-hard [6, 11].

All the above negative results show that the problem is very hard in terms of complexity. Nevertheless, Orman and Potts [7] showed that, three other special cases, i.e., $1 \mid \text{Coup-Task}, a_j = L_j = p \mid C_{\max}$, $1 \mid \text{Coup-Task}, b_j = L_j = p \mid C_{\max}$ and $1 \mid \text{Coup-Task}, a_j = b_j = b, L_j = l \mid C_{\max}$, are solvable in $O(n)$ time. They raised an open problem for the complexity of $1 \mid \text{Coup} - \text{Task}, a_j = a, L_j = l, b_j = b \mid C_{\max}$. Very recently, Ahr et al. [1] presented for this problem an exact algorithm using dynamic programming, which runs in $O\left(nr^{21}\right)$ time $\left(r < \sqrt[a-1]{a}\right)$. This algorithm is not polynomial in terms of *a* and *l*. Thus, the actual complexity of this problem still remains open.

While the general problem and its special cases have been analyzed in depth from a complexity point of view, analysis of algorithms for the problem is not much. Shapiro [10] gave three simple heuristics whose performance is analyzed based on experimental results. Leung and Zhao [6] showed that a greedy heuristic, which schedules the jobs in increasing order of L_j, is a 3-approximation algorithm for both the problem $1 \mid \text{Coup-Task}, a_j = a, b_j = b, a > b \mid C_{\max}$ and the problem $1 \mid \text{Coup-Task}, a_j = a, b_j = b, a < b \mid C_{\max}$. When $a = b$, the approximation bound is improved to 5/2. If $a = b = 1$, the approximation bound is further improved to 2. Recently, Ageev and Kononov [2] analyzed a 2.5-approximation algorithm for the case $1 \mid \text{Coup} - \text{Task}, a_j \le b_j \mid C_{\max}$ as well as a 3.5-approximation algorithm for the general case.

A related problem was studied by Kern and Nawijn [4], and Leung and Zhao [6] in which the delay between the operations of a job was assumed to be *minimum* instead of *exact*. They showed that the problem is NP-hard in the ordinary sense. Gupta [3] showed that this problem is strongly NP-hard, and compared several greedy heuristics by experimental approach. Reizebos, Gaalman, and Gupta [8] generalized the problem to multiple (not just two) operations per job and developed heuristics to solve it.

In this paper, we are interested in the design and analysis of algorithms for the coupled-task scheduling problem and its NP-hard special cases. For the NP-hard special cases, we present some algorithms that promise good approximation bounds. For the general case, while it is hard to obtain a good approximation algorithm, we design a tabu search meta-heuristic.

The remainder of the paper is organized as follows. In the next section, we give some preliminary results that will be used in later sections. In Section III, we present and analyze approximation algorithms for several NP-hard special cases.

Haibing Li is with Lehman Brothers Inc., New York City, NY 10019, USA. (hl27@njit.edu).

Hairong Zhao is with the Department of Mathematics, Computer Science & Statistics, Purdue University Calumet, Hammond, IN 46323, USA. (corresponding author to provide phone: 1-219-989-3181, e-mail: hairong@calumet.purdue.edu).

In Section IV, we develop a tabu search algorithm for the general problem. In Section V, we analyzed the performance of the algorithms using an experimental approach. Finally, we give some concluding remarks in Section VI.

II. Preliminaries

In this section, we give some definitions and present some preliminary results that will be used in later sections.

We first define two specific terms for this problem, namely *nesting* and *interleaving*. Given a schedule, a job *j* is said to be *nested* within job *k* if their tasks are finished in the order of a_k, a_j, b_j, b_k. A job *j* is said to be *interleaved* with job *k* if their tasks are finished in the order of a_j, a_k, b_j, b_k or in the order a_k, a_j, b_k, b_j. A schedule is a *permutation* schedule if no job is nested in another job. Usually, nesting and/or interleaving should be applied to obtain an optimal schedule for an instance of the problem.

Given an instance of our problem, a job *j* is called a *singleton* job if both nesting and interleaving are not possible for *j*. That is, for any other job *k*, *j* can not be nested within *k*, nor *k* can be nested within *j*, nor *j* can be interleaved with *k*. It is intuitive to observe the following property for an instance having *singleton* jobs.

Observation 1 (Singleton Job) *If an instance of the problem* $1|\text{Coup-Task}|C_{\max}$ *has singleton jobs, then an optimal schedule can be obtained by first finding the optimal schedule for the non-singleton jobs, then concatenating all singleton jobs one after another at the end of the schedule.*

As a result of the above observation, if in an instance every job is a singleton, then concatenating all the jobs in arbitrary order produces an optimal schedule.

Lemma 2 (Reverse Equivalence [7]) *The problem* $1|\text{Coup-Task},(a_j,L_j,b_j)|C_{\max}$ *and its reverse* $1|\text{Coup-Task},(b_j,L_j,a_j)|C_{\max}$ *are equivalent.*

Let *A*, *B* be the total processing time of the first and second operation of all jobs, respectively, i.e. $A=\sum_{j=1}^{n} a_j$, $B=\sum_{j=1}^{n} b_j$. We have the following lower bounds for the optimal makespan $C^*_{\max}$.

Lemma 3 (Lower bound) For any instance of the problem $1|\text{Coup-Task}|C_{\max}$, we have

$$C^*_{\max} \geq A+B, \qquad (1)$$

$$C^*_{\max} \geq \max_{1\le j\le n}\{a_j+L_j+b_j\}, \qquad (2)$$

$$C^*_{\max} \geq A+\min_{1\le j\le n}\{L_j\}, \qquad (3)$$

$$C^*_{\max} \geq B+\min_{1\le j\le n}\{L_j\}. \qquad (4)$$

Proof: The first bound holds since all the first and second operations have to be processed on the single machine. The second bound says that the makespan is at least the length of any job, in particular the longest job, which is obviously true. The third bound comes from the relaxed problem where $b_j=0$ for all *j*. And the last bound comes from the relaxed problem where $a_j=0$ for all *j*. □

III. Approximation Algorithms for NP-hard Special Cases

In this section, we focus on the design and analysis of some approximation algorithms for a variety of strongly NP-hard cases of the coupled-task scheduling problem.

The first strongly NP-hard case we consider is $1|\text{Coup-Task},a_j=b_j=L_j=p_j|C_{\max}$. We have the following observation for any schedule.

Observation 4 *In any schedule for an instance of* $1|\text{Coup-Task},a_j=b_j=L_j=p_j|C_{\max}$, *any two distinct jobs j and k can not be interleaved unless* $p_j=p_k$.

Given an instance of this problem, it is easy to see that the total length of the idle intervals of any schedule with no forced idle time is at most $L=\sum_{j=1}^{n}L_j=(A+B)/2$, which is at most $C^*_{\max}/2$ due to (1) in Lemma 3. Thus, we have

Lemma 5 *The makespan of any schedule with no forced idle time for* $1|\text{Coup-Task},a_j=b_j=L_j=p_j|C_{\max}$ *is at most 3/2 times the optimal.*

Now let us consider another strongly NP-hard case $1|\text{Coup}-\text{Task},b_j=b,L_j=l|C_{\max}$. We start with the following observation:

Observation 6 *No nesting is possible for the problem* $1|\text{Coup-Task},b_j=b,L_j=l|C_{\max}$. *In other words, for any instance of* $1|\text{Coup-Task},b_j=b,L_j=l|C_{\max}$, *every feasible schedule is a permutation schedule.*

Theorem 7 *One can find a schedule for the problem* $1|\text{Coup}-\text{Task},b_j=b,L_j=l|C_{\max}$ *in linear time whose makespan is at most 3 times the optimal.*

Proof: We consider 3 cases.

Case 1: $\max_{1\le j\le n}\{a_j\}\le b$, i.e. $a_j\le b$ for any $1\le j\le n$. We show the makespan of any schedule with no forced idle time is at most 2 times the optimal.

Let S be an arbitrary schedule with no forced idle time. Suppose the jobs are scheduled in the order of $1,2,\dots,n$.

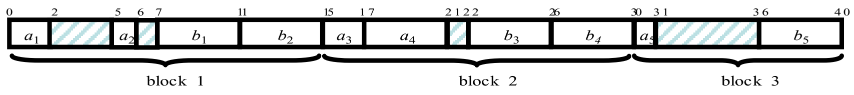

Fig. 1. Illustration of an arbitrary schedule of case1-instance in Theorem 7. The jobs are (2, 5, 4), (1, 5, 4), (2, 5, 4), (4, 5, 4), (1, 5, 4).

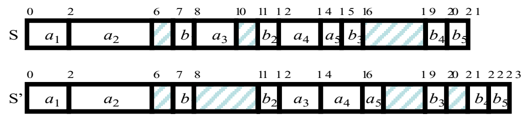
S
0 2 6 7 8 10 11 12 14 15 16 19 20 21
a_1 a_2 b_1 a_3 b_2 a_4 a_5 b_3 b_4 b_5
S'
0 2 6 7 8 11 12 14 16 19 20 21 22 23
a_1 a_2 b_1 b_2 a_3 a_4 a_5 b_3 b_4 b_5

Fig. 2. Illustration of an arbitrary schedule of case2-instance in Theorem 7. The jobs are (2, 5, 1), (4, 5, 1), (2, 5, 1), (2, 5, 1), (4, 5, 1).

Then it must be true that either a_{j+1} or b_{j+1} is scheduled immediately after b_j, see Fig. 1. for an illustration. To analyze the makespan of S, we divide the schedule into blocks so that if j and $j+1$ are in the same block, then b_{j+1} is scheduled immediately after b_j with no idle time. Suppose a block consists of jobs $i, i+1, ..., j$. Then the following must be true:

- By definition of block, b_i, b_{i+1}, ..., b_j are scheduled one after another with no idle time.
- Since j and $j+1$ are in different blocks, b_j is followed immediately by a_{j+1} with no idle time.
- For $i \le k < j-1$, we have $0 < a_{k+1} < b$ by assumption. The idle time after a_k is $0 \le b - a_{k+1} < b$.
- If $j \ne n$, the idle time after a_j is strictly less than b because otherwise $j+1$ will be in the same block as j; if $j = n$, the idle time after a_n is at most l.

Thus the total idle time is the sum of the lengths of those intervals after a_j, $1 \le j \le n$. From the above analysis, the length of the idle interval after a_j is at most b for $1 \le j \le n-1$ and the length of idle interval after a_n is at most l. Thus the total length of the idle intervals is at most $(n-1)b + l < B + l$. Therefore, the makespan of the schedule is at most $A + B + (B + l) < 2C^*_{\max}$ by Equation (1) and (4).

Case 2: $\min_{1 \le j \le n}\{a_j\} \ge b$, i.e, $a_j \ge b$ for any $1 \le j \le n$. We show any schedule with no forced idle time has makespan at most 2 times the optimal.

Let S be an arbitrary schedule with no forced idle time. Suppose the jobs area scheduled in the order of $1, 2, ..., n$. Now instead of analyzing the makespan of S, we analyze the makespan of a new scheduled S' that greedily schedules the jobs in the same order as S but subject to the condition that a_{j+1} is scheduled either immediately after a_j or immediately after b_j, see Fig. 2. for an illustration. With the additional constraint, it is obvious that S' has a larger makespan than that of S.

To analyze the makespan of S', we divide the schedule into blocks so that if j and $j+1$ are in the same block, then a_{j+1} is scheduled immediately after a_j. Suppose a block consists of jobs $i, i+1, ...j$. Then the following must be true:

- By definition of block, a_i, a_{i+1}, ... a_j are scheduled one after another with no idle time.
- Since j and $j+1$ are in different blocks, b_j is followed immediately by $a_j + 1$ with no idle time.
- For $i+1 \le k < j$, the length of the idle interval before b_k is $0 \le a_k - b < a_k$ since by assumption $0 < b \le a_k$.
- If i belongs to the last block, the idle time before b_i is at most $l_i = l$; otherwise the idle time before b_i is at most a_{j+1} since a_{j+1} can not fit into the interval.

Thus the total length of idle time is the sum of the lengths of those intervals before b_j. By the above analysis, the length of the idle interval before b_j is at most a_j, and the length of the idle interval before b_n is at most l. So the total length of idle intervals is at most $\sum_{i=1}^{n-1} a_j + l < A + l$. The makespan is at most $A + B + (A + l) \le 2C^*_{\max}$ by Equation (1) and (3).

Case 3: We divide the jobs into two groups: the first group contains jobs such that $a_j \le b$; the second group contains those jobs such that $a_j > b$. Arbitrarily schedule each group and concatenate them. Using A_1, B_1 to denote the sum of the first and second operations of the jobs in the first group, respectively. Similarly we define A_2 and B_2. The total makespan is at most:

$$\left(A_1 + B_1 + (B_1 + l)\right) + \left(A_2 + B_2 + (A_2 + l)\right)$$
$$= A + B + (B_1 + l) + (A_2 + l) < 3C^*_{\max},$$

due to the lower bounds in (1), (4) and (3).

Finally it is easy to see that our algorithm takes linear time in all cases. □

By Lemma 2 and Theorem 7, we immediately have the following corollary.

Corollary 8 *One can find a schedule for* $1 \mid \text{Coup}-\text{Task}, a_j = a, L_j = l \mid C_{\max}$ *in linear time whose makespan is at most 3 times the optimal.*

IV. A Meta-heuristic for the General Case

Needless to say, the general problem would be harder in terms of theoretical analysis of algorithm than the NP-hard special cases that we studied in the previous section. In this section, We develop a tabu search algorithm. For an introduction to tubu search, the reader is referred to Glover [13, 14]. Since a schedule for the general case is not necessarily a permutation schedule, this brings in some difficulties for generating neighborhoods for local searches, which would better be sequences. To resolve this problem, we use a heuristic, which is called *Construct-Schedule-from-Sequence* (*CSFS*), to construct a schedule from a given sequence of jobs $S = \langle j_1, j_2, ..., j_n \rangle$. Thus, it maps a sequence to a non-permutation schedule whose makespan can be determined accordingly. We describe the heuristic in detail as the following pseudo-code.

Algorithm $CSFS(S)$. Let S be the sequence $\langle j_1, j_2, ..., j_n \rangle$. Start job j_1 at time 0. For each index $i = 2, 3, ..., n$, scheldue job j_i at the ealiest possible time in the partial schedule. Return the makespan of the constructed schedule .

Essentially, to schedule a job at the earliest possible time in the partial schedule, the algorithm tries to apply nesting, interleaving, and appending in a greedy way. It should be noted that, combinations of different greedy sequencing rules with the above algorithm result in different heuristics. For example, we can order the jobs using the *SPT* rule or *LPT* rule on L_j, and feed the obtained sequence into the above heuristic. However, such fixed sequences could produce very bad schedules. Therefore, in what follows, we use a tabu search algorithm to explore the sequence space. Hopefully, this could find some good schedules in the search procedure.

Let us first consider a neighborhood generating mechanism, which is a necessity for the local search procedure. Given a sequence $S = \langle j_1, j_2, ..., j_n \rangle$ (accordingly implies a schedule mapped to by algorithm *CSFS*) and a parameter *K*, we generate a neighborhood using the following procedure:

Neighborhood (*S*, *K*). 1) Let the neighbohood be empty, i.e., $\mathcal{N} = \varnothing$. 2) For each position $i = 1, 2, ..., n$ in sequence S and each length $k = 1, 2, ..., K$, move the subseuqence $\langle j_i, j_{i+1}, ..., j_{i+k-1} \rangle$ to each position in the set $\langle 1, 2, ..., i-1, i+k, i+k+1, ..., n \rangle$, to obtain a new schedule S'. Add S' into $\mathcal{N}$. 3) Return $\mathcal{N}$ which contains all such new scheduleds constructed in step 2).

Now we are ready to give a tabu search algorithm. The algorithm use a problem instance *I* as feed, and starts search from a randomly generated sequence. When it terminates, a sequence mapped to a best schedule is returned. To prevent the algorithm from re-exploring those sequences that have been visited already, ideally we can store such explored sequences into memory as a tabu list. However, storing such sequences would require much memory. To resolve this problem, we map a sequence $S = \langle j_1, j_2, \ldots, j_n \rangle$ to a structure which is an ordered small sequence with only five items: $M(S) = \langle j_1, j_{\lceil n/3 \rceil}, j_{\lceil 2n/3 \rceil}, j_n, CSFS(S) \rangle$. We name this as tabu structure. With such a structure, we keep the tabu structure in memory in the life time of tabu search. If two different sequences share a same tabu structure, we simply treat two sequences as the same and assumes that they are mapped to the same schedule. If indeed there exists two different sequences share the same tabu structures, both will be prohibited from re-exploring. Thus, this tabu structure would prevent the local search from being trapped into local optima. We describe the tabu search algorithm as follows. In the algorithm, the notation is defined as follows:

$\mathcal{L}$: Tabu list;

S_b: Best sequence searched so far;

C_b: The makespan of the schedule constructed from S_b by $CSFS(S_b)$;

Algorithm ***TabuSearch***(I). 1) Randomly generate a sequence S for instance I. Set tabu list $\mathcal{L} = \varnothing$, $S_b = S$, $C_b = CSFS(S)$; Configure the value of K. 2) Search in $Neighborhood(S, K)$ the best neighbor S' such that $M(S')$ is not in the tabu list $\mathcal{L}$. If no such S' exists, randomly generate a new sequence S'. If $CSFS(S') < C_b$, let $S_b = S'$, $C_b = CSFS(S')$. 3) Let $\mathcal{L} = \mathcal{L} \cup \{M(S)\}$, $S = S'$, repeat 2) until the predefined stopping condition is met. 4) Return the schedule

TABLE I
THE AVERAGE OF $C_{\max}(H)$ TO C_{LB} AND AVERAGE RUNNING TIME OVER THE GENERATED INSTANCES

n	K	$\bar{r}(H)$			$\bar{s}(H)$ (in seconds)		
		ι =500	ι =1000	ι =5000	ι =500	ι =1000	ι =5000
50	3	1.193	1.155	1.134	0.004	0.027	0.089
	6	1.133	1.098	1.056	0.009	0.061	0.206
	10	1.115	1.051	1.049	0.013	0.362	0.753
100	3	1.387	1.295	1.216	0.015	0.059	0.271
	6	1.204	1.135	1.099	0.076	0.257	0.515
	10	1.187	1.130	1.112	0.252	0.713	1.972
200	3	1.452	1.335	1.308	0.077	0.174	0.489
	6	1.359	1.301	1.195	0.362	0.718	1.376
	10	1.320	1.267	1.209	0.750	1.233	3.572
500	3	1.538	1.502	1.411	0.290	0.981	3.745
	6	1.479	1.445	1.386	0.799	2.990	6.790
	10	1.408	1.358	1.339	0.750	5.119	12.843

constructed by $CSFS(S_b)$.

In the above algorithm, the stopping condition is configurable. A typical stopping condition could be a fixed number of non-improving iterations (denoted as ι). For example, if S_b has not been updated for 1000 iterations, then the algorithm terminates. We will use such stopping criterion in our experiments later.

V. COMPUTATIONAL RESULTS

To test our metaheuristic, we need some problem instances. Since no one has reported experimental results for the problem and no benchmark instances are available, we generate problem instances of different characteristics. For each $n = 50, 100, 200, 500$, we generate 20 instances. For each instance in each group, the a_j's, b_j's and L_j's are randomly chosen in the range $[1,100]$ subject to uniform distribution.

The algorithms are implemented in C++. The running environment is based on the Windows 2000 operating system; the PC used was a notebook computer (Pentium III 900Mhz plus 384MB RAM).

For configurable control parameters, in addition to the non-improving iterations ι which is set to 500, 1000, 2000, we also set $K = 3, 6, 10$.

Since it is unlikely that the optimal solution can be obtained by an exact algorithm very quickly, we compare the heuristic results with a lower bound of the optimal solution which can be computed easily using lemma 3. Let $C_{max}(H)$ be the metaheuristic result and C_{LB} be the lower bound for the optimal result which is denoated as C_{OPT}. We define

$$r(H) = \frac{C_{max}(H)}{C_{LB}}.$$

Obviously,

$$r(H) \geq \frac{C_{max}(H)}{C_{OPT}} \geq 1.$$

Thus, if $r(H)$ is close to 1, it means that the metaheuristic result is close to the lower bound. Hence, it would be even closer to the optimal cost. Therefore, to some extent, the ratio $r(H)$ indicates how good our metaheuristic is when it is applied to solve the problem instances.

For each instance generated, we run the tabu search algorithm with configured ι and K on it to produce a schedule. In addition, the value of C_{LB} is computed using lemma 3. With these, we compute $r(H)$ for this instance.

Due to the space constraints, we only show the statistic results obtained from the experiments. Table 1 shows the average of $r(H)$ and average running time (in seconds) on the 20 instances for each combination of $n = 50, 100, 200, 500$, $\iota = 500, 1000, 2000$, and $K = 3, 6, 10$. In the table, $\bar{r}(H)$ and $\bar{s}(H)$ denote the average ratio and average running time, respectively. From Table 1, we have the following observations:

- When n is small, K and ι are large, $\bar{r}(H)$ is close to 1. When n increases, $\bar{r}(H)$ also increases if K and ι are fixed. This implies that the search is more efficient for small n.
- When n and K are fixed, $\bar{r}(H)$ decreases when ι increases. This implies that the solution quality is dependent on the stopping criterion. However, the difference of the results between $\iota = 1000$ and $\iota = 2000$ is not too much.
- When n and ι are fixed, $\bar{r}(H)$ decreases when K increases. This implies that the solution quality is dependent on the neighborhood size. The larger the neighborhood size, the better the searched

schedule. However, the difference of the results between $K = 6$ and $K = 10$ is not too much.

- The running time is closely related to all n and K and ι, it increases if one of these parameters increases.

VI. Concluding Remarks

In this paper we analyzed approximation algorithms for some strongly NP-hard case of the coupled-task scheduling problem for the objective of minimizing makespan. We also proposed a tabu search approach for solving the general case, by exploring the structure of a valid schedule for the problem.

The tabu search algorithm was implemented to have experimental analysis. The observations on the experimental results reveal that they can produce in practice solutions that are very close to optimal with configurable parameters.

References

[1] D. Ahr, J. Békési, G. Galambos, M. Oswald and G. Reinelt, "An exact algorithm for scheduling identical coupled tasks," *Mathematical Methods of Operations Research*, vol.59, pp. 93–203, 2004.

[2] A. A. Ageev and A. V. Kononov, "Approximation Algorithms for Scheduling Problems with Exact Delays," *Working paper*, Sobolev Institute of Mathematics, Novosibirsk, Russia, 2006.

[3] J. N. D. Gupta, "Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags," *Journal of Global Optimization*, vol. 9, pp. 239–250, 1996.

[4] W. Kern, and W. M. Nawijn, "Scheduling multi-operation jobs with time lags on a single machine," *Proceedings 2nd Twente Workshop on Graphs and Combinatorial Optimization*, U. Faigle and C. Hoede (eds.), Enschede, 1991.

[5] J. K. Lenstra, Private communication, 1991.

[6] J.Y-T. Leung and H. Zhao, "Minimizing sum of completion times and makespan in master-slave systems," *IEEE Transactions on Computers*, vol. 55, pp. 985–999, 2006.

[7] A. J. Orman, and C. N. Potts, "On the complexity of coupled-task scheduling," *Discrete Applied Mathematics*, vol. 72, pp. 141–154, 1997.

[8] J. Riezebos, G. Gaalman, and J. N. D. Gupta, "Flowshop scheduling with multiple operations and time-lags," *Journal of Intelligent Manufacturing*, vol. 6, pp. 105–115, 1995.

[9] A. H. G. Rinnooy Kan, Machine Scheduling Problems, Martinus Nijhoff, The Hague.

[10] R. D. Shapiro, "Scheduling Coupled Tasks," *Naval Research Logistics Quarterly*, vol. 20, pp. 489–498, 1980.

[11] W. Yu, H. Hoogeveen and J. K. Lenstra, "Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard," *Journal of Scheduling*, vol.7, pp. 333–348, 2004.

[12] F.Glover, "Tabu Search - Part I," *ORSA Journal on Computing*, vol. 1, pp. 190–206, 1989.

[13] F.Glover, "Tabu Search - Part II," *ORSA Journal on Computing*, vol. 2, pp. 4–32, 1990.