Discrete Applied Mathematics 160 (2012) 2370–2388

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Scheduling coupled-operation jobs with exact time-lags

A. Condotta, N.V. Shakhlevich *

School of Computing, University of Leeds, Leeds, LS2 9JT, United Kingdom

ARTICLE INFO

Article history:
Received 10 June 2011
Received in revised form 1 March 2012
Accepted 29 May 2012
Available online 11 July 2012

Keywords:
Coupled-operation scheduling
Exact time-lags
NP-hardness
Tabu search

ABSTRACT

Scheduling coupled-operation jobs with exact time-lags on a single machine has a wide range of applications. In that problem, each job consists of two operations with given processing times, which should be scheduled on a single machine observing a given time-lag. The general case of the problem with arbitrary processing times of operations and arbitrary time lags is known to be NP-hard in the strong sense and the problem remains NP-hard for many special cases. In order to develop a local search algorithm for the problem, we first explore two possible approaches for representing feasible solutions and their neighborhoods based on maintaining a permutation of first operations of the jobs or maintaining a full permutation of all operations. The first representation appears to be unpromising since, as we prove, the problem of finding an optimal sequence of second operations for a fixed sequence of first operations is NP-hard in the strong sense even in the case of unit processing times. We elaborate the second approach by developing a tabu search heuristic based on efficient job re-insertion. Empirical evaluation demonstrates superiority of the developed algorithm in comparison with the earlier published algorithms.

1. Introduction

Coupled-operation scheduling has been the subject of intensive research since the late 90s. In a typical problem of this type each job consists of two operations which should be processed one after another with a pre-specified time-lag in-between. During the delay interval, the machine may process operations of other jobs. The objective is to sequence all operations on one machine so that the makespan is minimum.

The coupled-operation problem arises in various applications: radar controllers receive signals after a delay since the pulse emission [10,11,23,25]; some health care treatments must follow delivery patterns with strictly defined time-lags [9]; in distributed computing the master processor organizes data transmission and the time-lags in-between transmissions of input and output files correspond to the execution stage by a slave processor [18,19].

The coupled-operation problem is known to be NP-hard and theoretical aspects of complexity analysis have received considerable attention of the researchers. Publications on solution techniques are mainly focused on approximation algorithms and ILP-based methods for real-world applications [10,25]. The most systematic study of heuristics for the coupled-operation scheduling problem is presented in [22] where the authors establish a number of important properties of the problem and develop several construction heuristics and local search algorithms.

Formally, in the model under consideration each job j of the set $N = \{1, 2, \ldots, n\}$ consists of a pair of operations a_j and b_j which should be processed without preemption by a single machine. Job j is characterized by a triple p_{a_j}, L_j, p_{b_j}, where p_{a_j}

* Corresponding author. Fax: +44 133 343 5444.
E-mail addresses: alessandro@condotta.net (A. Condotta), n.shakhlevich@leeds.ac.uk (N.V. Shakhlevich).

0166-218X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2012.05.026

and p_{b_j} are processing times of a_j and b_j and L_j is the duration of a given time-lag. If the first operation a_j starts at time s_{a_j}, then the second operation b_j should start exactly at time

$$s_{b_j} = s_{a_j} + p_{a_j} + L_j. \tag{1}$$

The machine can be used for processing other jobs in-between a_j and b_j, but the time-lag of duration L_j between completing a_j and starting b_j should be observed. The completion time of the second operation determines the completion time c_j of job j,

$$c_j = s_{b_j} + p_{b_j}.$$

The objective is to sequence all $2n$ operations of jobs N so that the machine processes at most one operation at a time and the makespan $C_{\max} = \max_{j\in N}\{c_j\}$ is minimum. Extending the notation from [12,21] we denote the problem by $1 \mid a_j, b_j, L_j \mid C_{\max}$.

The introduced model is characterized by exact time-lags: in accordance with (1), the difference $s_{b_j} - \left(s_{a_j} + p_{a_j}\right)$ between the starting time of b_j and completion time of a_j should be exactly equal to L_j. There are several related models of coupled-operation scheduling in which time-lags are treated differently. In the model with flexible time-lags, denoted by $1 \mid a_j, b_j, \ell_j, u_j \mid C_{\max}$, the difference between the starting time of b_j and completion time of a_j should be within given boundaries,

$$\ell_j \le s_{b_j} - \left(s_{a_j} + p_{a_j}\right) \le u_j. \tag{2}$$

In the model with minimum time-lags, denoted by $1 \mid a_j, b_j, \ell_j, u_j = \infty \mid C_{\max}$, the upper bound is unlimited so that the time-lag between the first and the second operations can be indefinitely large.

All formulated versions of the coupled-operation problem are strongly NP-hard, see [21,24,26]. Polynomial-time algorithms are known only for special cases and most of the results for the problem with exact time-lags are formulated in [21] where a detailed classification of NP-hard and polynomially solvable cases is given. The long-standing open question on the complexity of problem $1 \mid a_j = b_j = p, L_j = L \mid C_{\max}$ has been recently resolved in [5], where the $O(\log n)$-time dynamic programming algorithm is proposed improving the preliminary results from [3]. The most recent research is mainly focused on various special cases of the problem with equal job parameters (e.g., equal processing times), but in a more general setting: coupled-operation jobs are replaced by chains of several operations, see [8,20].

As far as approximation algorithms are concerned, the problem with exact time-lags $1 \mid a_j, b_j, L_j \mid C_{\max}$ is approximable within a factor of $7/4$ in the case of unit-time operations $a_j = b_j = 1$ [1], within a factor of $5/2$ in the case of equal-length operations $a_j = b_j$ and within a factor of $7/2$ in the general case (arbitrary a_j and b_j) [2]; it is also shown in [2] that for the latter problem no polynomial-time algorithm exists with an approximation ratio $2 - \varepsilon$ unless $P = NP$.

Another stream of research considers coupled-operation problems with precedence constraints, see, e.g., the paper by Blazewicz et al. [6]. Even if precedence constraints define a complete sequence π of all operations, finding the optimum starting times is a non-trivial task if time-lags are flexible. The algorithm involves the longest path calculation in a disjunctive graph having positive- and negative-weight arcs (see [17,22]). Notice that the complexity status of the coupled-operation problem with a given sequence π_a of first operations or with a given sequence π_b of second operations is unknown.

To the best of our knowledge, there is only one publication [22] which discusses heuristic algorithms for the coupled-operation problem. It studies the version of the problem with flexible time-lags $1 \mid a_j, b_j, \ell_j, u_j \mid C_{\max}$ comparing different construction heuristics and local search algorithms. Interestingly, the proposed local search algorithms appear to be less efficient than the most successful construction heuristics and such inefficiency is explained by a high cost of generating and evaluating infeasible solutions. Indeed, the algorithms developed in [22] for flexible time-lags are applicable to the version with exact time-lags $1 \mid a_j, b_j, L_j \mid C_{\max}$. However, due to the importance of the latter problem for real-world applications, it is particularly desirable to design efficient problem-specific algorithms for it. As we show in our study, this can be achieved by exploiting special properties of the problem.

Our main objective is to design a successful local search method. We start our study with the analysis of possible representations of feasible solutions and the neighborhood structure. The local search approach from [22] developed for the problem $1 \mid a_j, b_j, \ell_j, u_j \mid C_{\max}$ with flexible time-lags represents feasible solutions as permutations π of all $2n$ operations and generates neighbors by removing one operation from π and inserting it elsewhere. The quality of a new solution is evaluated via the $O\left(n^2\right)$ longest path algorithm which either fixes the starting times of all operations observing given bounds ℓ_j, u_j for time-lags or identifies that no feasible solution exists. As noted in [22], the resulting local search approach performs poorly in comparison with construction heuristics since infeasible solutions prevail around local optima. Moreover, the performance is even worse for instances with no flexibility (i.e., for problem $1 \mid a_j, b_j, L_j \mid C_{\max}$ with exact time-lags) as infeasible neighbors occur more often for non-adjustable time-lags. Due to this reason, in our study we pay special attention to alternative representations of feasible solutions and alternative neighborhood structures.

For solution representation, one natural approach is based on considering permutation π_a of a-operations or permutation π_b of b-operations. Notice that due to the problem symmetry for the makespan objective, a problem with a fixed permutation π_a can be re-formulated as the problem with a fixed permutation π_b. Using a single permutation π_a (or permutation π_b) may seem attractive as this representation is compact and leads to simple strategies for neighbor generation. Such a representation induces an important subproblem in which the permutation π_a (or π_b) is fixed and the objective is to produce a complete schedule, i.e., to specify complete permutation π of all $2n$ operations and their starting times so that

the makespan $C_{\max}$ is minimum. We perform complexity analysis of that subproblem in Section 2 and demonstrate that it is NP-hard in the strong sense even in the case of unit processing time. This negative result suggests that using representation π_a (or, equivalently, π_b) is less preferable in comparison with the full permutation π.

As far as neighbor generation strategy is concerned, we take into account the conclusion from [22] on inefficiency of re-insertion of a single operation in a given permutation π of $2n$ operations as it often leads to infeasible permutations. The alternative neighbor generating strategy we suggest optimally re-inserts the whole job consisting of two operations. It adapts a so-called short cycle theory developed in [13–15] for job re-insertion in job-shop models of special types.

Although the strategy of job re-insertion may look very similar to operation re-insertion from [22], the new strategy has a dramatic effect on the performance of the local search algorithm: it always generates feasible solutions during the search and the neighbors are obtained as solutions to a specially defined optimization problem. Enumerating neighbors of good quality is perhaps one of the reasons of good performance of our method.

The formulated ideas are elaborated in the tabu search algorithm presented in Section 3. We suggest two enhancements that improve the search:

- maintaining the pool of solutions ranked in accordance with the estimates of possible improvements that can be achieved if a neighbor is generated;
- creating the tabu list which keeps the main characteristics of the eliminated solutions in the format of critical paths.

The performance of the tabu search algorithm is evaluated empirically comparing it with the winning method from [22]—the randomized construction heuristic. Its adaptation for the model with exact time-lags is described in Section 4 followed by the summary of computational experiments in Section 5. Conclusions are drawn in Section 6.

2. NP-hardness of the problem with a given sequence of first operations

It is known that if the permutation π of $2n$ operations is given, then their optimum starting times can be found in $O\left(n^2\right)$ time as a solution to a specially defined longest path problem [22]. In this section we consider the related problem in which the permutation π_a of a-operations is given while permutation of b-operations is not fixed. Without loss of generality we assume that the jobs are numbered in accordance with the permutation of a-operations so that $\pi_a = (1, 2, \ldots, n)$. The objective is to find the starting times of all operations and a complete permutation π of all operations so that the makespan $C_{\max}$ is minimum.

Our main result is the proof that the coupled-operation problem with a given permutation π_a of a-operations is NP-hard in the strong sense even if all operations have unit processing times. Using the 3-field notation, the problem is denoted by $1|a_j = b_j = 1, L_j, \pi_a|C_{\max}$. The decision version of this problem consists in verifying whether there exists a feasible solution with the makespan not-exceeding a given threshold value T. To simplify the notation, the latter problem is denoted by COET (π_a, T), where COET stays for Coupled-Operation problem with Exact Time-lags.

We reduce the following Coupled-Operation problem with Minimum Time-lags and a given makespan threshold t, denoted by COMT(t), to problem COET (π_a, T).

Problem COMT(t): given a set of jobs $Q = \{1, 2, \ldots, q\}$, each job $j \in Q$ consisting of two unit-time operations separated by a time-lag of duration no less than ℓ_j, does there exist a feasible schedule with the makespan no larger than t? A schedule is feasible if no two operations are processed simultaneously and the second operation of each job j starts after at least ℓ_j time units elapse since the first operation of that job is completed. It is known that COMT(t) is NP-complete in the strong sense, see [26].

Given an instance $\mathcal{I}(t)$ of problem COMT(t), we construct an instance $\mathcal{I}'(\pi_a, T)$ of problem COET (π_a, T) as follows. The set of jobs N in instance $\mathcal{I}'(\pi_a, T)$ consists of $n = 3tq+1$ coupled-operation jobs. Their time-lags L_j are defined in accordance with the job type.

- There are q sets $F_1, \ldots, F_q$ of the so-called *filler jobs*, each set $F_h = \{(3t-2)(h-1)+1, \ldots, (3t-2)h\}$, $1 \le h \le q$, consisting of $3t-2$ jobs with time-lags
$$L_j = 3tq, \quad j \in F_h.$$
- There is one set R consisting of a single *sentinel job*, $R = \{(3t-2)\,q+1\}$, with time-lag
$$L_j = t(3q+1), \quad j \in R.$$
- There are q sets $U_1, \ldots, U_q$ of U-jobs, each set $U_h = \{(3t-2)q+2h\}$, $1 \le h \le q$, consisting of a single job with time-lag
$$L_j = 3t\,(q-h) + 2t - 1, \quad j \in U_h.$$
- There are q sets $V_1, \ldots, V_q$ of V-jobs, each set $V_h = \{(3t-2)q+2h+1\}$, $1 \le h \le q$, consisting of a single job with time-lag
$$L_j = 3t\,(q-h) + 2t - 1 + \ell_h, \quad j \in V_h,$$
where ℓ_h is the minimum time-lag from instance $\mathcal{I}(t)$ of problem COMT(t).

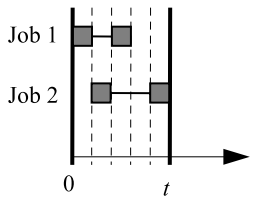

Fig. 1. An example of a feasible schedule for instance $\mathcal{I}(t)$ of problem COMT(t) with $t = 5$ and two coupled-operation jobs with minimum time-lags $\ell_1 = 1$ and $\ell_2 = 2$.

Observe that

$$L_j \geq 2t - 1 \quad \text{for } j \in U_h \cup V_h. \tag{3}$$

The makespan threshold value is $T = 6tq + t + 2$ and the sequence of a-operations is $\pi_a = (1, 2, \ldots, 3tq + 1)$:

$$\pi_a = (\underbrace{1, \ldots, 3t-2}_{F_1}, \underbrace{3t-1, \ldots, 2\,(3t-2)}_{F_2}, \ldots, \underbrace{(3t-2)\,(q-1)+1, \ldots, (3t-2)\,q}_{F_q},$$
$$\underbrace{(3t-2)\,q+1}_{R}, \underbrace{(3t-2)q+2}_{U_1}, \underbrace{(3t-2)q+3}_{V_1}, \underbrace{(3t-2)q+4}_{U_2}, \underbrace{(3t-2)q+5}_{V_2}, \ldots, \underbrace{3tq}_{U_q}, \underbrace{3tq+1}_{V_q}). \tag{4}$$

An example of a feasible schedule for instance $\mathcal{I}(t)$ of problem COMT(t) with makespan $t = 5$ and two coupled-operation jobs ($q = 2$) with minimum time-lags $\ell_1 = 1$ and $\ell_2 = 2$ is shown in Fig. 1. The corresponding feasible schedule for instance $\mathcal{I}'(\pi_a, T)$ of problem COET(π_a, T) with $T = 6tq + t + 2 = 67$ is shown in Fig. 2.

In what follows, we use the following notation for single-element sets:

$$R = \{r\},$$
$$U_h = \{u(h)\},$$
$$V_h = \{v(h)\}.$$

Theorem 1. *If there exists a feasible schedule for instance $\mathcal{I}(t)$ of problem* COMT(t)*, then there exists a feasible schedule for instance $\mathcal{I}'(\pi_a, T)$ of problem* COET (π_a, T).

Proof. Let a feasible solution S for instance $\mathcal{I}(t)$ be given by starting times s_{a_h} and s_{b_h}, $h \in Q$, of its a- and b-operations. We construct a feasible schedule S' for instance $\mathcal{I}'(\pi_a, T)$ with permutation π_a given by (4). First we describe the structure of that schedule and then specify the starting times s'_{a_j} and s'_{b_j}, $j \in N$, for all operations.

There are five types of time intervals:

- q time intervals $\lambda_1, \lambda_2, \ldots, \lambda_q$, each of length $3t$, such that in interval
$$\lambda_h = [3t\,(h-1)\,, 3th], \quad 1 \leq h \leq q,$$
all first operations of the jobs from F_h are processed; since there are $3t - 2$ jobs in F_h and the length of interval λ_h is $3t$, there are two idle time intervals of unit length in each interval λ_h;
- one unit-length time interval
$$\xi = [3tq, 3tq + 1]$$
for processing the first operation of the sentinel job from R;
- q time intervals $\mu_1, \mu_2, \ldots, \mu_q$, each of length $3t$, such that in interval
$$\mu_h = [3t\,(q + h - 1) + 1, 3t\,(q + h) + 1], \quad 1 \leq h \leq q,$$
all second operations of the jobs from F_h are processed together with the first operations of the jobs from U_h and V_h; since there are $3t - 2$ jobs in F_h, one job in U_h and one job in V_h, there are no idle time slots in μ_h;
- one interval τ of length t defined as
$$\tau = [6tq + 1, 6tq + 1 + t]$$
for processing the second operations of the jobs from $\cup_{h=1}^{q} U_h$ and $\cup_{h=1}^{q} V_h$;
- one unit-length time interval
$$\eta = [6tq + t + 1, 6tq + t + 2]$$
for processing the second operation of the sentinel job from R.

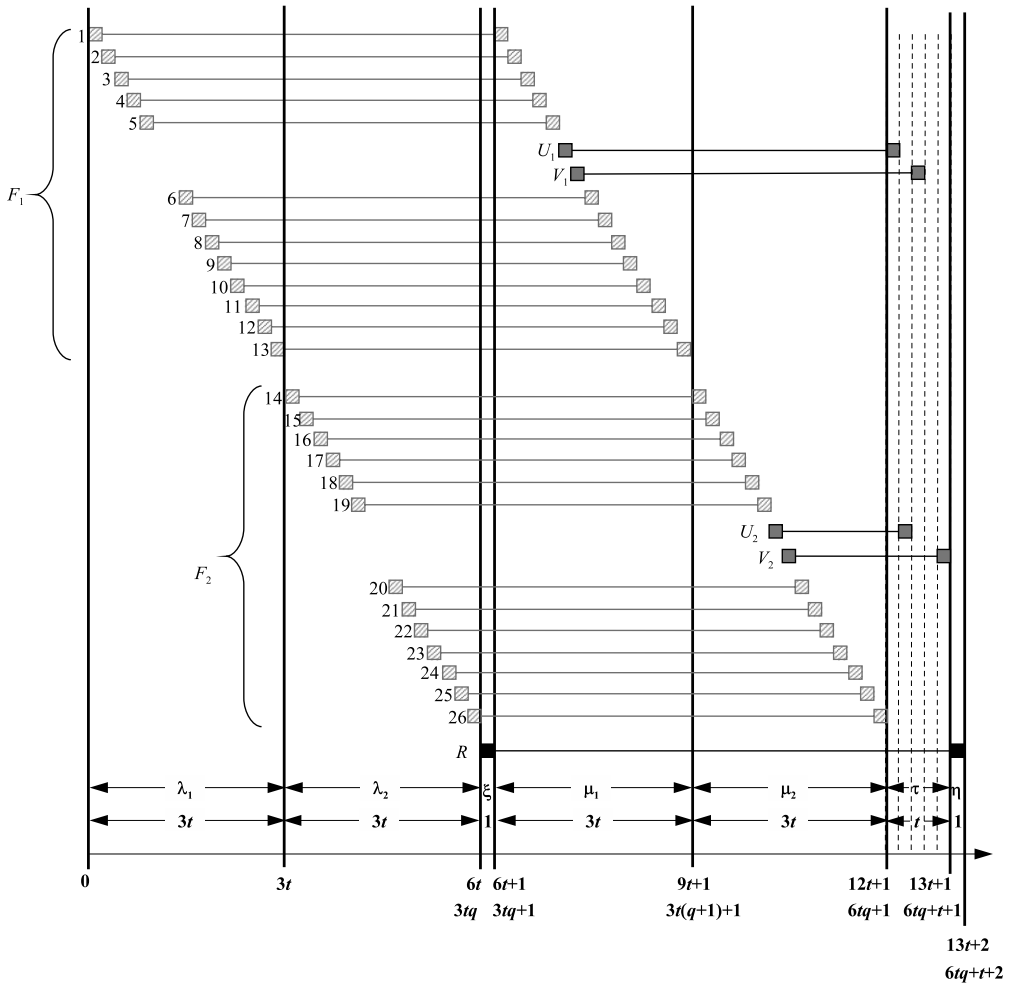

Fig. 2. An example of a feasible schedule for instance COET $(\pi_a, 67)$ generated from a feasible schedule for problem COMT(5) with two coupled-operation jobs with minimum time-lags $\ell_1 = 1$ and $\ell_2 = 2$.

To give a full description of the schedule, we specify the starting times $s'_{b_1}, s'_{b_2}, \ldots, s'_{b_n}$, of the b-operations of jobs N; the starting times of the corresponding a-operations are then derived as $s'_{a_j} = s'_{b_j} - L_j - 1$.

The job completing at time T in the schedule for instance $\mathcal{I}'(\pi_a, T)$ is the sentinel job $r \in R$ and its two operations are processed in time intervals ξ and η:

$$s'_{b_r} = T - 1 = 6tq + t + 1, \qquad s'_{a_r} = (6tq + t + 1) - t\,(3q + 1) - 1 = 3tq.$$

Scanning the schedule for instance $\mathcal{I}'(\pi_a, T)$ from its right end backwards, we define the starting times of the jobs from $V_q, U_q, V_{q-1}, U_{q-1}, \ldots, V_1, U_1$. In the expressions below, $s'_{a_{v(h)}}, s'_{b_{v(h)}}$ and $s'_{a_{u(h)}}, s'_{b_{u(h)}}$ are the starting times of the operations of jobs $v(h)$ and $u(h)$, $1 \le h \le q$, for instance $\mathcal{I}'(\pi_a, T)$ and they are calculated on the basis of the starting times s_{a_h} and s_{b_h} of the two operations of job $h \in Q$, $1 \le h \le q$, in a feasible schedule for instance $\mathcal{I}(t)$:

$$\begin{aligned}
s'_{b_{v(h)}} &= 6tq + 1 + s_{b_h}, & s'_{a_{v(h)}} &= \left(6tq + 1 + s_{b_h}\right) - (3t\,(q-h) + 2t - 1 + \ell_h) - 1 \\
&& &= 3t\,(q+h) + s_{b_h} - 2t + 1 - \ell_h, \\
s'_{b_{u(h)}} &= 6tq + 1 + s_{a_h}, & s'_{a_{u(h)}} &= \left(6tq + 1 + s_{a_h}\right) - (3t\,(q-h) + 2t - 1) - 1 \\
&& &= 3t\,(q+h) + s_{a_h} - 2t + 1.
\end{aligned} \tag{5}$$

Observe that the partial schedule for $\mathcal{I}'(\pi_a, T)$ in the interval τ involves the second operations of jobs $u(h)$ and $v(h)$, $1 \le h \le q$, and it coincides with the full schedule of length t for instance $\mathcal{I}(t)$:

$$\begin{aligned}
s'_{b_{u(h)}} &= (6tq + 1) + s_{a_h}, \\
s'_{b_{v(h)}} &= (6tq + 1) + s_{b_h}.
\end{aligned}$$

As far as the first operations of jobs $u(h)$ and $v(h)$ are concerned, they satisfy the following properties:

(i) for each h, $1 \leq h \leq q$, the first operations of $u(h)$ and $v(h)$ are both processed in time interval μ_h;
(ii) the first operation of job $v(h)$ starts after the first operation of job $u(h)$:

$$s'_{a_{v(h)}} \geq s'_{a_{u(h)}} + 1.$$

We now define the time intervals for the filling jobs. Consider the set F_h for some h, $1 \leq h \leq q$. The $3t-2$ jobs in F_h have their second operations processed in time interval μ_h. The length of the latter interval is $3t$ and two unit-time slots within μ_h are already fixed for $u(h)$ and $v(h)$, see (5). Thus there is a unique way to allocate the jobs from F_h in time interval μ_h keeping them in the order of their numbering and avoiding clashes with $u(h)$ and $v(h)$.

Having allocated the second operations of the jobs from F_h in interval μ_h, their first operations are automatically allocated in interval λ_h. Since the time-lags for the jobs in F_h are equal, the order of the first operations is the same as the order of the second operations and it satisfies the given permutation π_a. Taking into account observation (i), the order given by π_a is satisfied for all jobs. □

In the remaining part we prove that if there exists a feasible schedule for instance $\mathcal{I}'(\pi_a, T)$ of problem COET (π_a, T), then there exists a feasible schedule for the related instance $\mathcal{I}(t)$ of problem COMT(t). We assume that t satisfies the following two conditions:

$$t \geq \ell_j + 2, \quad j \in Q, \tag{6}$$

$$t \geq 2q; \tag{7}$$

otherwise COMT(t) does not have a feasible solution since the makespan of the coupled-operation schedule cannot be smaller either of the values, the length of a single job or the combined length of q jobs consisting of two unit-time operations.

We start with a lemma which characterizes the structure of a feasible solution for instance $\mathcal{I}'(\pi_a, T)$.

Lemma 1. *In any feasible solution S' for instance $\mathcal{I}'(\pi_a, T)$ of problem* COET (π_a, T) *the following properties hold:*

1) *the a-operations of all jobs from N are processed in one of the time intervals $\lambda_1, \ldots, \lambda_q, \xi, \mu_1, \ldots, \mu_q$, i.e.,*

$$s'_{a_j} \leq 6tq, \quad j \in N;$$

2) *the a-operation of the sentinel job $r \in R$ is processed in time interval ξ or earlier, i.e.,*

$$s'_{a_r} \leq 3tq;$$

3) *the a-operations of all jobs from $\cup_{h=1}^{q} F_h$ are processed in time intervals $\lambda_1, \ldots, \lambda_q$, i.e.,*

$$s'_{a_j} \leq 3tq - 1, \quad j \in F_h;$$

4) *the b-operations of all jobs from F_h, $1 \leq h \leq q$, are processed in time intervals $\mu_1, \ldots, \mu_q$, i.e.,*

$$3tq + 1 \leq s'_{b_j} \leq 6tq, \quad j \in F_h;$$

5) *each unit time slot of intervals $\mu_1, \ldots, \mu_q$ has an operation allocated to it; the set of operations allocated to intervals $\mu_1, \ldots, \mu_q$ consists of all b-operations of the jobs from $\cup_{h=1}^{q} F_h$ and exactly one operation of each job from $\cup_{h=1}^{q} (U_h \cup V_h)$.*
6) *job $f = (3t-2)q$, which is the last job of the set F_q, starts at time $3tq - 1$;*
7) *the a-operations of all jobs from $\cup_{h=1}^{q} (U_h \cup V_h)$ are processed in time intervals $\mu_1, \ldots, \mu_q$, i.e.*

$$s'_{a_j} \geq 3tq + 1, \quad j \in \cup_{h=1}^{q} (U_h \cup V_h);$$

8) *the b-operations of all jobs from $\cup_{h=1}^{q} (U_h \cup V_h)$ are processed in time interval τ, i.e.*

$$6tq + 1 \leq s'_{b_j} \leq 6tq + t, \quad j \in \cup_{h=1}^{q} (U_h \cup V_h).$$

Proof.

1) For any job j from $\cup_{h=1}^{q} (U_h \cup V_h)$, its time-lag satisfies inequality (3), while all other jobs have even larger time-lags. For any job $j \in N$, its completion time should not be larger than $T = 6tq + t + 2$, which implies that

$$s'_{a_j} \leq T - L_j - 2 \leq (6tq + t + 2) - (2t - 1) - 2 = 6tq - t + 1 \leq 6tq.$$

2) To be completed within the threshold makespan value T, the starting time of the a-operation of job $r \in R$ should satisfy

$$s'_{a_r} \leq T - L_r - 2 = (6tq + t + 2) - t(3q + 1) - 2 = 3tq.$$

3) In accordance with sequence π_a, any filler job from $\cup_{h=1}^{q} F_h$ must start before the sentinel job from R, i.e., at time $3tq-1$ or earlier.

4) By the previous property, all filler jobs from $\cup_{h=1}^{q} F_h$ have their starting times within $[0, 3tq-1]$. Since all of them have the same time-lag of $3tq$, the b-operations start within time interval $[3tq+1, 6tq]$.

5) Due to the property 4) of the current lemma, there are $(3t-2)q$ b-operations of the jobs from $\cup_{h=1}^{q} F_h$ which have their starting times in $[3tq+1, 6tq]$, leaving room for at most $2q$ additional unit-time operations in $\mu_1, \ldots, \mu_q$. We show that exactly $2q$ operations of the jobs from $\cup_{h=1}^{q} U_h$ and $\cup_{h=1}^{q} V_h$, are scheduled in $\mu_1, \ldots, \mu_q$, one operation of each job.

Consider a job $j \in \cup_{h=1}^{q} (U_h \cup V_h)$. If

$$s'_{a_j} \geq 3tq+1, \tag{8}$$

then due to property 1), the a-operation of job j is processed within $\mu_1, \ldots, \mu_q$.

For the alternative case with

$$s'_{a_j} \leq 3tq, \tag{9}$$

we show that the corresponding b-operation is processed within $\mu_1, \ldots, \mu_q$.

To prove that the left boundary $3tq+1$ of μ_1 is observed, we derive the lower bounds on s'_{a_j} and s'_{b_j}. Since the set of jobs which precede j in permutation π_a includes $(3t-2)\,q$ operations of filler jobs from $\cup_{h=1}^{q} F_h$ and one sentinel job from R,

$$s'_{a_j} \geq (3t-2)\,q+1.$$

Taking into account condition (3), we obtain:

$$s'_{b_j} = s'_{a_j} + L_j + 1 \geq ((3t-2)\,q+1) + (2t-1) + 1 = 3tq + 2\,(t-q) + 1 \geq 3tq + 2q + 1,$$

where the last inequality is due to assumption (7).

To prove that the right boundary $6tq$ of μ_q is observed, we derive the upper bound on s'_{b_j} using condition (9) and the fact that the time-lag L_j is bounded by the maximum time-lag in the set $\cup_{h=1}^{q} (U_h \cup V_h)$, which is $L_{v(1)} = 3t\,(q-1) + 2t - 1 + \ell_1$. We conclude:

$$s'_{b_j} = s'_{a_j} + L_j + 1 \leq 3tq + (3t\,(q-1) + 2t - 1 + \ell_1) + 1 = 6tq - t + \ell_1 \leq 6tq - 2,$$

where the last inequality is due to assumption (6).

Thus in any case, (8) or (9), at least one operation of job $j \in \cup_{h=1}^{q} (U_h \cup V_h)$ is processed within intervals $\mu_1, \ldots, \mu_q$. Taking into account that there are $2q$ unit time intervals left after allocation of the b-operations of $\cup_{h=1}^{q} F_h$, we conclude that exactly one operation of every job j is processed within intervals $\mu_1, \ldots, \mu_q$.

6) Consider job $f = (3t-2)q$, which is the last one in the set F_q. By property 3) of this lemma,

$$s'_{a_f} \leq 3tq - 1.$$

To prove that

$$s'_{a_f} \geq 3tq - 1$$

we show that

$$s'_{b_f} \geq 6tq.$$

Suppose that $s'_{b_f} < 6tq$. By property 5), there is a job $j \in \cup_{h=1}^{q} (U_h \cup V_h)$ allocated to time interval $[6tq, 6tq+1]$. If it is the first operation of j, i.e., $s'_{a_j} = 6tq$, then the deadline T is violated:

$$s'_{b_j} = s'_{a_j} + L_j + 2 \geq 6tq + (2t-1) + 2 > T,$$

where the first inequality is due to (3). If it is the second operation of j, i.e.,

$$s'_{b_j} = 6tq,$$

then job j would have two operations processed within $\mu_1, \ldots, \mu_q$ since its time-lag satisfies

$$L_j \leq 3t\,(q-1) + 2t - 1 + \ell_h \leq 3tq - 3,$$

which implies

$$s'_{a_j} = s'_{b_j} - L_j - 1 \geq 6tq - (3tq-3) - 1 = 3tq + 2,$$

a contradiction to property 5) of this lemma.

7) In accordance with permutation π_a, the starting time of the last job of the set F_q with index $f = (3t-2)q$ is smaller than that of the sentinel job $r \in R$ which in its turn is smaller than the starting time of any job $j \in \cup_{h=1}^{q}(U_h \cup V_h)$:

$$s'_{a_f} < s'_{a_r} < s'_{a_j},$$

so that

$$s'_{a_j} \geq s'_{a_f} + 2.$$

The statement now follows from property 6).
8) The total length of time intervals $\mu_1, \ldots, \mu_q$ is $3tq$ and in accordance with properties 5) and 7) of this lemma there are exactly $3tq$ operations allocated there: $(3t-2)\,q$ operations from $\cup_{h=1}^{q} F_h$ and $2q$ a-operations from $\cup_{h=1}^{q}(U_h \cup V_h)$. Therefore all b-operations of the jobs from $\cup_{h=1}^{q}(U_h \cup V_h)$ are processed after time intervals $\mu_1, \ldots, \mu_q$.

It remains to show that no b-operation from $\cup_{h=1}^{q}(U_h \cup V_h)$ can be processed in time interval η. If it is a case, then for the sentinel job $r \in R$,

$$s'_{b_r} \leq 6tq + t,$$
$$s'_{a_r} = s'_{b_r} - L_r - 1 = s'_{b_r} - t\,(3q+1) - 1 \leq 3tq - 1.$$

The latter condition cannot happen since, due to property 6), the a-operation of the last job $f = (3t-2)\,q$ from F_q is assigned to time slot $[3tq-1, 3tq]$ and it should precede the a-operation of the sentinel job. □

Based on the properties formulated in Lemma 1, we now prove Theorem 2 which, in combination with Theorem 1, implies NP-completeness of problem COET (π_a, T).

Theorem 2. *If there exists a feasible schedule for instance $\mathcal{I}'(\pi_a, T)$ of problem* COET (π_a, T)*, then there exists a feasible schedule for instance $\mathcal{I}(t)$ of problem* COMT(t).

Proof. Consider a feasible schedule for instance $\mathcal{I}'(\pi_a, T)$. By construction of instances $\mathcal{I}(t)$ and $\mathcal{I}'(\pi_a, T)$, for any job h of instance $\mathcal{I}(t)$, its two operations a_h and b_h correspond to the following two operations of instance $\mathcal{I}'(\pi_a, T)$: the b-operation of job $u(h) \in U_h$ and the b-operation of job $v(h) \in V_h$.

Due to property 8) of Lemma 1, all b-operations of the jobs from $\cup_{h=1}^{q}(U_h \cup V_h)$ are processed in time interval τ of length t. We define a solution to instance $\mathcal{I}(t)$ via the partial solution to instance $\mathcal{I}'(\pi_a, T)$ in the time interval τ. For any job h, $1 \leq h \leq q$, of instance $\mathcal{I}(t)$, the starting times of its two operations are defined as

$$s_{a_h} := s'_{b_{u(h)}} - (6tq+1)\,, \quad u(h) \in U_h,$$
$$s_{b_h} := s'_{b_{v(h)}} - (6tq+1)\,, \quad v(h) \in V_h.$$

It remains to show that in instance $\mathcal{I}(t)$

$$s_{b_h} \geq s_{a_h} + 1 + \ell_h,$$

or equivalently

$$s'_{b_{v(h)}} \geq s'_{b_{u(h)}} + 1 + \ell_h.$$

Indeed, due to permutation π_a which is observed in the solution to instance $\mathcal{I}'(\pi_a, T)$,

$$s'_{a_{v(h)}} \geq s'_{a_{u(h)}} + 1.$$

Taking into account that

$$L_{u(h)} = 3t\,(q-h) + 2t - 1,$$
$$L_{v(h)} = 3t\,(q-h) + 2t - 1 + \ell_h,$$

we conclude:

$$\begin{aligned} s'_{b_{v(h)}} - s'_{b_{u(h)}} &= \left[s'_{a_{v(h)}} + L_{v(h)} + 1\right] - \left[s'_{a_{u(h)}} + L_{u(h)} + 1\right] \\ &\geq \left[s'_{a_{u(h)}} + 1 + (3t\,(q-h) + 2t - 1 + \ell_h)\right] - \left[s'_{a_{u(h)}} + (3t\,(q-h) + 2t - 1)\right] \\ &\geq \ell_h + 1. \quad \square \end{aligned}$$

The NP-hardness result proved in this section holds also for the symmetric counterpart of the problem in which permutation π_b of b-operations is fixed, i.e., for $1|a_j = b_j = 1, L_j, \pi_b|C_{\max}$. Indeed, any instance of problem $1|a_j = b_j = 1, L_j, \pi_a|C_{\max}$ with $\pi_a = (1, 2, \ldots, n)$ can be converted into an equivalent instance of problem $1|a_j = b_j = 1, L_j, \pi_b|C_{\max}$ by swapping a- and b-operations of each job and renumbering the jobs in the reverse order so that for the resulting instance

permutation $\pi_b = (1, 2, \ldots, n)$ is fixed. For that pair of instances, there exists a one-to-one correspondence between pairs of feasible schedules: if s_{a_j}, s_{b_j}, $j \in N$, are the starting times for one problem, then $C_{\max} - s_{b_j} - 1$ and $C_{\max} - s_{a_j} - 1$ are the starting times for its counterpart.

Observe that problem $1|a_j = b_j = 1, L_j, \pi_a|C_{\max}$ considered in this section is a special case of the problem with equal processing times $1|a_j = b_j = p, L_j, \pi_a|C_{\max}$ which in its turn is a special case of a more general problem $1|a_j, b_j, L_j, \pi_a|C_{\max}$. This fact implies that the latter two problems are also NP-hard in the strong sense, as well as their counterparts with the fixed sequence π_b.

3. Tabu search algorithm

One of the key decisions in the design of a tabu search algorithm is the choice of the solution representation and neighbor generation strategy. The representation based on the fixed permutation of a-operations (or b-operations) seems to be attractive as it is compact and supports well studied neighbor generation rules. However, as shown in the previous section, that representation incurs an NP-hard problem of sequencing all operations and finding job starting times minimizing the makespan. Due to this reason we use an alternative representation given by permutation π of all $2n$ operations. We suggest a neighbor generation strategy based on removing both operations of a selected job j and inserting them elsewhere so that the makespan is minimized and the sequence of the remaining $2(n-1)$ operations is unchanged. An efficient insertion algorithm for finding the optimum re-allocation of job j is based on the disjunctive graph model (see, e.g., [7]) and the short cycle property formulated in [13–15] for the job shop environment.

We start with describing the disjunctive graph model and the insertion algorithm (Section 3.1). Then we describe the neighborhood structure (Section 3.2). Finally, we conclude with the general description of the tabu search algorithm (Section 3.3).

3.1. Disjunctive graph model and the insertion algorithm

In this section we present an efficient procedure for solving the problem of inserting two operations of a given job j into a partial schedule defined for $n-1$ jobs $N \setminus \{j\}$. We assume that the sequence of operations of the jobs $N \setminus \{j\}$ is fixed and an upper bound $C^{UB}_{\max}$ on the makespan value is given; the objective is to find a complete feasible schedule, if one exists, with job j inserted such that the order of operations from $N \setminus \{j\}$ is not altered and the upper bound $C^{UB}_{\max}$ is not exceeded.

The algorithm is based on the disjunctive graph model which can be described as follows. A disjunctive graph $G = (V, A, E, \mathcal{E}, c)$ is a directed graph with the vertex-set V, the set of conjunctive arcs A, the set of disjunctive arcs E, the family $\mathcal{E} \subseteq 2^E$ of disjunctive sets and the arc cost function c.

Set V contains vertices v_i, $i = 1, 2, \ldots, 2n$, representing $2n$ operations of the coupled-operation problem and two additional dummy nodes: origin v_0 and terminal v_*. If v_i represents the a-operation (b-operation) of job j, notation a_j (b_j, respectively) is used in line with v_i.

Conjunctive arcs A and the costs associated with them, set restrictions on the minimum difference in the starting times of the operations they connect: if there is an arc (v_i, v_j), then the starting times s_{v_i} and s_{v_j} should satisfy

$$s_{v_j} \geq s_{v_i} + c_{v_i v_j}.$$

There are four conjunctive arcs for each job $j \in N$:

- two arcs (a_j, b_j) and (b_j, a_j) represent precedence constraints between operations of the same job; their costs $c_{a_j b_j} = p_{a_j} + L_j$ and $c_{b_j a_j} = -(p_{a_j} + L_j)$ specify the exact distance between the starting times of a_j and b_j;
- one arc (v_0, a_j) from origin v_0 to the a-operation of job j has a zero cost $c_{v_0 a_j} = 0$;
- one arc (b_j, v_*) from the b-operation of job j to the terminal node v_* has the cost $c_{b_j v_*} = p_{b_j}$.

One additional conjunctive arc (v_*, v_0) connects the terminal node and the origin; its cost is based on the given value of the makespan upper bound: $c_{v_* v_0} = -C^{UB}_{\max}$, which implies

$$s_{v_0} \geq s_{v_*} - C^{UB}_{\max}$$

or equivalently

$$C_{\max} = s_{v_*} \leq s_{v_0} + C^{UB}_{\max}.$$

Set E contains pairs $(e, \bar{e})$ of disjunctive arcs $e = (v_i, v_j)$ and $\bar{e} = (v_j, v_i)$ which connect two operations of different jobs. We call $\bar{e}$ the mate of e and vice versa. In a complete solution, one of the disjunctive arcs of the pair is selected specifying the order between the two operations while the other arc is dropped. The cost of a disjunctive arc (v_i, v_j) is defined as the processing time of the operation corresponding to node v_i.

The family of disjunctive sets $\mathcal{E} \subseteq 2^E$ has the meaning that for each pair $(e, \bar{e})$, where $e = (v_i, v_j)$ and $\bar{e} = (v_j, v_i)$, at least one of the disjunctive constraints should be satisfied:

$$s_{v_j} \geq s_{v_i} + c_{v_i v_j} \quad \text{or} \quad s_{v_i} \geq s_{v_j} + c_{v_j v_i}.$$

An example of a disjunctive graph for three coupled-operation jobs is shown in Fig. 3.

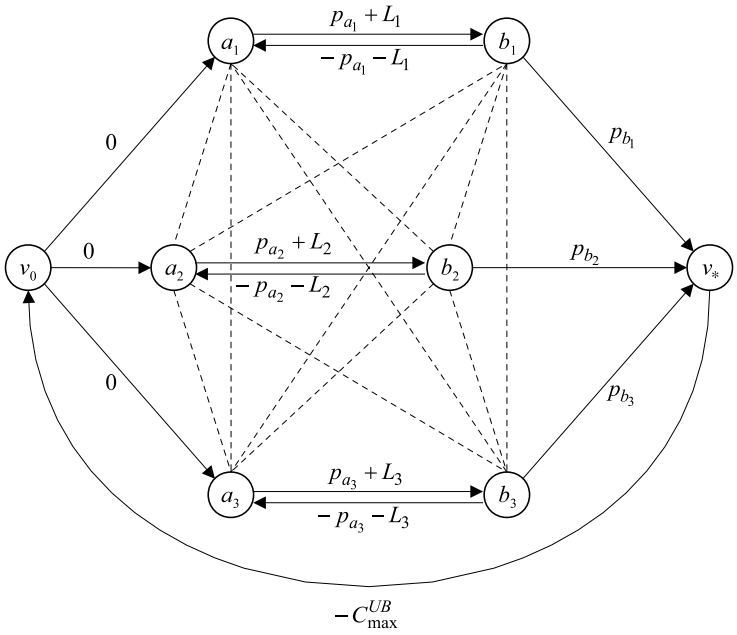

Fig. 3. Disjunctive graph for three coupled-operation jobs (pairs of disjunctive arcs are represented by dashed lines).

The scheduling decision consists in finding a *selection* of disjunctive arcs $\mathcal{F} \subseteq E$ containing no more than one mate from a pair. Selection $\mathcal{F}$ is *feasible* if it is *complete* (exactly one arc is selected from each pair of arcs $(e, \bar{e})$) and *positive acyclic* (digraph $(V, A \cup \mathcal{F}, c)$ does not contain cycles of positive cost). A feasible selection $\mathcal{F}$ specifies a schedule $\mathcal{S}$ in which the starting time of every operation v_i is defined as the length of the longest path from origin v_0 to the corresponding node v_i in digraph $(V, A \cup \mathcal{F}, c)$. The makespan of the schedule is equal to the length of the *critical path*, i.e., the longest directed path from the origin v_0 to the terminal node v_*. Notice that for an infeasible selection the critical path does not exist since it would have an infinite cost; for a feasible selection a critical path can be found in $O\left(|V|^2\right)$ time.

We can now introduce the neighbor generation problem for the coupled-operation problem. It is defined for the current partial solution $\mathcal{S}$ corresponding to selection $\mathcal{F}$ and job $j \in N$ to be inserted in $\mathcal{S}$. We assume that $\mathcal{F}$ is complete and positive acyclic for the subproblem defined by the nodes $V \setminus \{a_j, b_j\}$; otherwise inserting two operations of job j cannot lead to a feasible solution. For the insertion set $J = \{a_j, b_j\}$ we define the set of disjunctive arcs $E(J) \subseteq E$ and the set of conjunctive arcs $A(J) \subseteq A$ which connect nodes in J and the remaining nodes $V \setminus J$. Notice that conjunctive arcs $\left(a_j, b_j\right)$ and $\left(b_j, a_j\right)$ are not included in $A(J)$.

Definition 1. Given a job j with operations $J = \{a_j, b_j\}$ and a selection $\mathcal{F}$ feasible for the nodes $V \setminus J$ and not containing any disjunctive arcs from $E(J)$, the *job insertion problem* consists in finding a selection $\mathcal{F}^J \subset E(J)$ such that the combined selection $\mathcal{F} \cup \mathcal{F}^J$ is feasible:

- $\left|\mathcal{F}^J\right| = \frac{1}{2}|E(J)|$,
- the resulting graph $G = (V, A \cup \mathcal{F} \cup \mathcal{F}^J, c)$ is positive acyclic.

The associated *insertion graph* is denoted by $G^J = (V, A \cup \mathcal{F}, E(J), \mathcal{E}(E(J)), c)$, where $\mathcal{E}(E(J)) \subset \mathcal{E}$ is the family of disjunctive sets defined over $E(J)$.

A feasible selection can be found efficiently by the procedure from [13] if insertion graph G^J has a so called *short cycle property*. It states that for any selection $\mathcal{F}^J$, if the corresponding digraph $(V, A \cup \mathcal{F} \cup \mathcal{F}^J, c)$ contains a positive cycle, then it contains a short positive cycle visiting J exactly once. Here 'a cycle visits J exactly once' has the meaning that exactly one arc of the cycle has the origin outside J and the end-node a_j or b_j, and exactly one arc has the start node a_j or b_j and the end-node outside J. We first demonstrate that this property is satisfied for the insertion graph of the coupled-operation problem and then describe how the algorithm from [13] can be applied to solving our insertion problem.

Proposition 1. *An insertion graph G^J of the coupled-operation problem has the short-cycle property.*

Proof. Suppose $\mathcal{F}^J$ is a positive cyclic selection with more than two arcs from $E(J)$. We show how a short positive cycle visiting J exactly once can be constructed.

Let Z be a positive cycle with arcs from $A \cup A(J) \cup \mathcal{F}^J$ which visits J twice or more. The possible components of cycle Z are arcs $\left(a_j, b_j\right)$, $\left(b_j, a_j\right)$ which may appear multiple times in Z and the fragments of the form a_jPa_j, b_jQb_j, a_jRb_j and b_jTa_j, where P, Q, R and T are the paths consisting of nodes different from $\{a_j, b_j\}$.

If Z has a fragment of the form a_jPa_j (or b_jQb_j) and such a fragment is of positive length, then it is a required short positive cycle; if that fragment is of non-positive length, it can be removed from Z so that the remaining cycle is positive and it has less visits to J.

If Z has a fragment of the form a_jRb_j and its length $c(R)$ satisfies

$$c(R) > c_{a_jb_j}, \tag{10}$$

then a combination of that fragment with arc (b_j, a_j) is a short positive cycle $a_jRb_ja_j$: it visits J only once and its length $c(R) + c_{b_ja_j}$ is positive since $c_{b_ja_j} = -c_{a_jb_j}$. Alternatively, if condition (10) does not hold, we replace fragment a_jRb_j of cycle Z by a single arc $\left(a_j, b_j\right)$ obtaining a new cycle with less visits to J and which length is positive since $c(R) \leq c_{a_jb_j}$.

It is easy to make sure that a similar transformation can be done for fragment b_jTa_j: one has to swap a_j and b_j in the above arguments and replace condition (10) by $c(T) > c_{b_ja_j}$.

Thus considering the fragments of Z with end nodes in $\left\{a_j, b_j\right\}$ we either construct a short positive cycle based on that fragment or eliminate a part of cycle Z resulting in a new positive cycle with less arcs and less visits to J. Applying this procedure we eventually produce a short positive cycle with exactly one visit to J. □

The short cycle property implies that a selection is feasible if and only if it does not include disjunctive arcs of the following two types:

(i) arc $e \in E(J)$ which incurs a positive cycle in the digraph $(V, A \cup \mathcal{F} \cup \{e\}, c)$;
(ii) a combination of two arcs $u, v \in E(J)$ which incur a positive cycle in the digraph $(V, A \cup \mathcal{F} \cup \{u, v\}, c)$.

This restriction on the choice of disjunctive arcs can be modeled naturally as a *conflict graph* [13]. Formally, a conflict graph $H_{G^J} = (X, U)$ has nodes $X = E(J)$ corresponding to disjunctive arcs of the insertion graph G^J and undirected arcs U of two types:

- loops $(e, e) \in U$ for disjunctive arcs e of type (i),
- undirected arcs $(u, v) \in U$ for every pair of disjunctive arcs u, v of type (ii).

Then any positive acyclic selection corresponds to a stable set in conflict graph H_{G^J}, and the required feasible selection $\mathcal{F}^J$ for graph G^J is the stable set of maximal cardinality in H_{G^J}, $\left|\mathcal{F}^J\right| = \frac{1}{2}|E(J)|$. If the maximum cardinality of a stable set is less than $\frac{1}{2}|E(J)|$, then no feasible solution exists.

Due to Proposition 1, H_{G^J} is bipartite with node partition $X = E^+(J) \cup E^-(J)$, where $E^+(J)$ are the arcs of the set $E(J)$ outgoing from J and $E^-(J)$ are the arcs incoming to J. Clearly, if the insertion graph G^J does not have disjunctive arcs of type (i), then a stable set of cardinality $\frac{1}{2}|E(J)|$ can be selected as $E^+(J)$ or $E^-(J)$. In the presence of disjunctive arcs of type (i), finding a stable set is not so straightforward; still it can be found efficiently, as suggested in [13], without employing standard approach based on the minimum cut problem.

Disjunctive arcs of type (i) dictate some limitations on the choice of selection $\mathcal{F}^J$: arc e of type (i) cannot be included in $\mathcal{F}^J$ or equivalently the corresponding node e of the conflict graph cannot be included in the stable set. Hence for each e of type (i) we should include its mate $\bar{e}$ in the stable set. Having included $\bar{e}$ in the stable set, all nodes $f \in E(J)$ connected with $\bar{e}$ by arcs $(\bar{e}, f)$ in the conflict graph cannot be included in the stable set since arcs $\bar{e}, f$ are of type (ii). Prohibiting f implies that its mate $\bar{f}$ should be selected and the procedure should be continued for $\bar{f}$.

Having performed all "necessary" selections dictated by disjunctive arcs of type (i), the selected nodes and their mates can be excluded from further consideration. The resulting subgraph $\widehat{H}_{G^J} \subseteq H_{G^J}$ is bipartite and it contains only the nodes of type (ii). A stable set for $\widehat{H}_{G^J}$ can be selected as one of the two bipartition sets.

The correctness of the described approach for the insertion problem with the short cycle property is rigorously proved in [13] and it can be formally presented as the following procedure.

Procedure 'Insert(G^J, J)'

Given: job j with operations $J = \{a_j, b_j\}$;
selection $\mathcal{F}$ feasible for the nodes $V \setminus J$ and not containing any disjunctive arcs from $E(J)$
Objective: find feasible selection $\mathcal{F}^J \subset E(J)$ such that $\left|\mathcal{F}^J\right| = \frac{1}{2}|E(J)|$ and the resulting graph $G = (V, A \cup \mathcal{F} \cup \mathcal{F}^J, c)$ is positive acyclic

1. Construct the bipartite conflict graph $H_{G^J} = (X, U)$; identify all arcs e of type (i) and include their mates $Q = \{\bar{e} \in E : (e, e) \in U\}$ in $\mathcal{F}^J$.
2. For every $u \in Q$ find all nodes $\left\{\bar{f}_1, \bar{f}_2, \ldots, \bar{f}_k\right\}$ reachable in H_{G^J} from u through alternating paths of the form $\left(u, f_1, \bar{f}_1, f_2, \bar{f}_2, \ldots, f_k, \bar{f}_k\right)$; denote the set of nodes reachable from Q by Q^*, $Q \subseteq Q^*$, and extend $\mathcal{F}^J$ by including Q^*.
3. If Q^* is not stable, then terminate: no feasible selection exists.
4. Otherwise, consider a subgraph $\widehat{H}_{G^J} = (\widehat{X}, \widehat{U})$ of H_{G^J} by eliminating all nodes Q^*, their mates and associated arcs; define a stable set T in $\widehat{H}_{G^J}$ as $\widehat{X} \cap E^+(J)$ or $\widehat{X} \cap E^-(J)$.
   Extend $\mathcal{F}^J$ by including T.

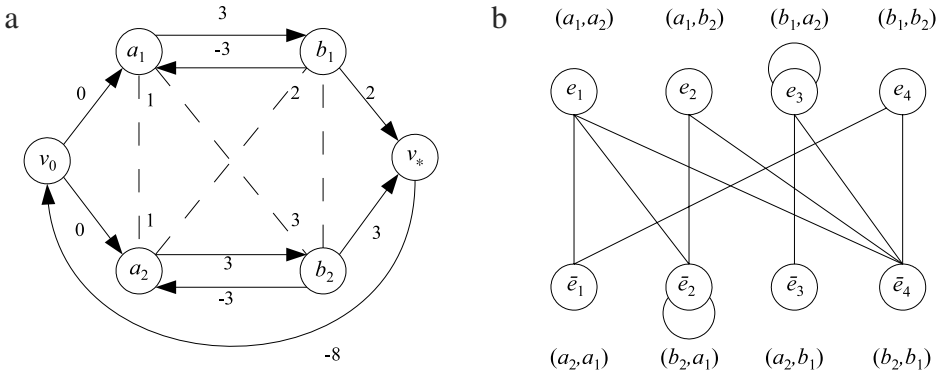

Fig. 4. (a) Disjunctive graph $G(V, E)$ and (b) conflict graph H_{G^J} for insertion set $J = \{a_1, b_1\}$.

Table 1
Short positive cycles of the graph in Fig. 4(a).

Positive cycle	Cycle value	Disjunctive arcs
$c_1 = (a_1, b_1, b_2, a_2)$	3	$(a_2, a_1), (b_1, b_2)$
$c_2 = (a_1, a_2, b_2, b_1)$	4	$(a_1, a_2), (b_2, b_1)$
$c_3 = (a_1, a_2, b_2)$	7	$(a_1, a_2), (b_2, a_1)$
$c_4 = (a_1, b_2, b_1)$	1	$(a_1, b_2), (b_2, b_1)$
$c_5 = (a_2, b_2, b_1)$	8	$(b_1, a_2), (b_2, b_1)$
$c_6 = (v_0, a_2, b_2, a_1, b_1, v_*)$	3	(b_2, a_1)
$c_7 = (v_0, a_1, b_1, a_2, b_2, v_*)$	3	(b_1, a_2)

In the case of the coupled-operation scheduling problem, conflict graph H_{G^J} has $|X| = 8(n-1)$ nodes corresponding to $2(n-1)$ pairs of disjunctive arcs connecting a_j with the remaining $2(n-1)$ nodes and similarly $2(n-1)$ pairs of disjunctive arcs incident to b_j. Implementation of Step 1 requires $O(n^3)$ time: it incurs the all-pairs longest path algorithm for preprocessing (e.g., the Floyd–Warshall algorithm, see [4]) and then it checks each disjunctive arc of G^J for property (i) and each combination of two disjunctive arcs for property (ii). Steps 2 and 3 can be implemented in $O(n^2)$ time, while Step 4 requires $O(n)$ time. Thus the overall time complexity of the described procedure is $O(n^3)$.

Observe that having introduced the arc (v_*, v_0) of weight $-C^{UB}_{\max}$ in the disjunctive graph model, we guarantee that procedure 'Insert' finds a feasible schedule (if one exists) with the makespan no larger than $C^{UB}_{\max}$. In order to find a feasible insertion minimizing the makespan, one can apply procedure 'Insert' in combination with binary search with trial makespan values in the range $\left[C^{LB}_{\max}, C^{UB}_{\max}\right]$ defined by the lower and upper bound of the makespan. One algorithm for calculating lower bounds is discussed in Section 5.2.

Example. Consider an instance with two jobs with processing times $p_{a_1} = 1, p_{b_1} = 2, p_{a_2} = 1, p_{b_2} = 3$, time-lags $L_1 = 2$ and $L_2 = 2$ and maximum makespan value $C^{UB}_{\max} = 8$. The corresponding disjunctive graph $G(V, E)$ is shown in Fig. 4(a). The costs of disjunctive arcs are placed next to the nodes they originate from. For the insertion set $J = \{a_1, b_1\}$, the conflict graph H_{G^J} is shown in Fig. 4(b). All short positive cycles are listed in Table 1.

Notice that procedure 'Insert(G^J, J)' finds a feasible insertion of $J = \{a_1, b_1\}$ by fixing disjunctive arcs $\mathcal{F}^J = \{e_1, e_2, \bar{e}_3, e_4\}$.

3.2. Neighborhood structure

Given a current complete schedule $\mathcal{S}$ specified by a feasible selection $\mathcal{F}$, we define the neighborhood of $\mathcal{S}$ via removing one of the critical jobs from that solution and inserting it optimally without altering the order of other operations. Job j is *critical* if operation a_j, operation b_j or both operations belong to the critical path in the disjunctive graph.

Unlike the earlier approach from [22], instead of re-inserting a chosen critical operation, we re-insert a critical job consisting of two operations ensuring that the resulting solution is feasible and has the minimum makespan. In order to guarantee that the neighbor differs from $\mathcal{S}$, we force one of the disjunctive arcs e belonging to the critical path in $\mathcal{S}$ to be replaced by its mate $\bar{e}$ oriented in the opposite direction. We call e a *disjunctive critical arc* and we select e to be incident to an operation of the critical job chosen for re-insertion. Formally, the neighbor generating procedure can be described as follows.

Procedure '*Neighbor*$(\mathcal{S}, j, e)$'

Given: current solution $\mathcal{S} = (V, A \cup \mathcal{F}, c)$;

selected critical job j;
selected disjunctive critical arc $e \in \mathcal{F}$ incident to j
Objective: generate a neighbor of $\mathcal{S}$ by re-inserting j and replacing e by its mate $\bar{e}$

1. Define the insertion set $J = \{a_j, b_j\}$ and the insertion graph G^J by removing all disjunctive arcs $E(J)$ incident to operations a_j and b_j from $\mathcal{F}$:
$$G^J = (V, A \cup (\mathcal{F} \setminus E(J)), E(J), \mathcal{E}(E(J)), c).$$
2. Include the mate $\bar{e}$ of the given disjunctive critical arc e into the selection and apply procedure 'Insert(G^J, J)' in combination with binary search to find a new feasible selection $(\mathcal{F} \setminus E(J)) \cup \{\bar{e}\} \cup \mathcal{F}^J$ for which the makespan value is minimum. Denote the resulting schedule by $\mathcal{S}'_{j,e}$.
3. Return schedule $\mathcal{S}'_{j,e}$ as the neighbor of $\mathcal{S}$.

Observe that procedure 'Insert(G^J, J)' presented in the previous section does not guarantee that the required arc $\bar{e}$ is selected. However, this can be easily ensured if a loop $(e, e) \in U$ is included in the conflict graph H_{G^J} so that Procedure 'Insert' is forced to select $\bar{e}$.

In what follows we are mainly interested in the best possible neighbor $\mathcal{S}'_j$ of the current solution $\mathcal{S}$ obtained via re-inserting job j. In order to find $\mathcal{S}'_j$, we need to identify all disjunctive critical arcs $E' \subset \mathcal{F}$ incident to j, apply procedure 'Neighbor($\mathcal{S}, j, e$)' for each arc $e \in E'$ and select the neighbor $\mathcal{S}'_j$ with the smallest makespan value:

$$C_{\max}(\mathcal{S}'_j) = \min_{e \in E'} \{C_{\max}(\mathcal{S}'_{j,e})\}.$$

Notice that the set E' contains two disjunctive arcs if job j has one operation on the critical path, or four disjunctive arcs, if both operations of job j belong to the critical path.

Consider now the problem of finding a neighbor of the current solution $\mathcal{S}$ having the smallest possible makespan. The straightforward approach consists in generating neighbor schedules $\mathcal{S}'_j$ for all critical jobs j and finding the required neighbor with the smallest value $C_{\max}(\mathcal{S}'_j)$. This, however, is computationally expensive due to the high computational cost of the relatively slow procedure 'Insert' combined with binary search.

A possible alternative approach is to calculate some estimates $\sigma(\mathcal{S}'_j)$ of the makespan value of each neighbor schedule $\mathcal{S}'_j$ assuming that a potentially good neighbor has the lowest estimate. For a fast method, we consider only Step 1 of Procedure 'Neighbor' without taking into account the selected disjunctive critical arc e and the subsequent 'Insert' procedure. The resulting estimate can be used as a quality measure of the insertion graph G^J. The most accurate characteristic of G^J is its makespan, which can be found in $O(n^2)$ time via the longest path calculation:

$$\sigma_1(\mathcal{S}'_j) = C_{\max}(G^J). \tag{11}$$

A faster but less accurate characterization of G^J can be found in $O(1)$ time as

$$\sigma_2(\mathcal{S}'_j) = C_{\max}(\mathcal{S}) - \Delta, \tag{12}$$

where Δ is a possible decrease in the makespan that can be achieved if job j is removed from $(\mathcal{S})$,

$$\Delta = \begin{cases} p_{a_j} + L_j + p_{b_j}, & \text{if operations } a_j \text{ and } b_j \text{ appear in the critical path in the order } a_j, b_j, \\ p_{b_j}, & \text{if operations } a_j \text{ and } b_j \text{ appear in the critical path in the order } b_j, a_j, \\ p_{a_j}, & \text{if only one operation } a_j \text{ appears in the critical path,} \\ p_{b_j}, & \text{if only one operation } b_j \text{ appears in the critical path.} \end{cases}$$

Notice that

$$C_{\max}(G^J) \le C_{\max}(\mathcal{S}'_j)$$

since inserting job j in G^J can only increase $C_{\max}(G^J)$. Therefore estimate (11) can be considered as the lower bound on the value of $C_{\max}(\mathcal{S}'_j)$,

$$\sigma_1(\mathcal{S}'_j) \le C_{\max}(\mathcal{S}'_j).$$

However, estimate (12) may satisfy

$$\sigma_2(\mathcal{S}'_j) \le C_{\max}(\mathcal{S}'_j) \quad \text{or} \quad \sigma_2(\mathcal{S}'_j) \ge C_{\max}(\mathcal{S}'_j)$$

since either condition

$$\sigma_2(\mathcal{S}'_j) \le C_{\max}(G^J) \quad \text{or} \quad \sigma_2(\mathcal{S}'_j) \ge C_{\max}(G^J)$$

may hold.

Empirical evaluation of the described techniques has demonstrated that generating all neighbor schedules of the current schedule $\mathcal{S}$ and calculating their makespan values is not efficient since it slows down the search procedure. Having performed experiments with two types of estimates we conclude that calculating σ_2 is much faster in comparison with σ_1 and leads to better results as a wider part of the solution space is examined fast enough. Therefore the approach adopted in computational experiments generates only one potentially good neighbor on the basis of the estimate σ_2.

3.3. Description of the tabu search implementation

In this section we describe implementation details of our tabu search algorithm. The algorithm maintains *tabu list* $\mathcal{T}$ of the characteristics of the most recent visited solutions to prevent reproducing those solutions and to avoid generating solutions with similar characteristics, see [16] for the general description of the tabu search algorithm.

The visited solutions are represented in the tabu list by their critical paths. For a critical path recorded in the list, we keep two additional characteristics:

- a *hit counter* which specifies how many solutions with that critical path were visited,
- a *serial number* of the last visited solution with that critical path.

Once a new neighbor is generated, its critical path is added to the tabu list, if it does not duplicate an existing entry in the tabu list; otherwise a hit counter of the existing entry is incremented and a serial number is updated.

Whenever the number of entries in the tabu list reaches a given threshold on the list size, the list is halved by removing the entries with the smallest hit counter values (i.e., solutions visited less often); in case of ties, the entries with the smallest serial numbers (i.e., the older solutions) are eliminated first. In the updated list, the hit counters are set to zero for all entries. This elimination strategy guarantees that the tabu list contains critical paths of the solutions which are visited most frequently during the search and, in addition, generated most recently.

While a traditional tabu search algorithm considers one current solution at a time and performs moves from one current solution to its neighbor, in our implementation we maintain a pool $\mathcal{Q}$ of current solutions, which have unvisited neighbors. Initially, the pool is filled in with solutions generated by the construction heuristics, which we describe in Section 4. Each time a solution is generated, we find estimates of all its neighbors. As discussed in Section 3.2, we use estimate σ_2 to find characteristics of neighbors fast enough. In each iteration we consider solutions of pool $\mathcal{Q}$, select an unvisited neighbor with the minimum estimate and generate it. The newly generated neighbor is either rejected, if its critical path is in the tabu list, or, otherwise, it is accepted and included in pool $\mathcal{Q}$. Whenever the number of entries in pool $\mathcal{Q}$ reaches a given threshold value, $\mathcal{Q}$ is emptied and refilled with a given number of schedules generated by construction heuristics.

Since it is important to have a fast procedure to recognize whether a neighbor has been previously examined or not, we maintain for each schedule $\mathcal{S} \in \mathcal{Q}$ the list $\chi(\mathcal{S})$ of critical jobs of that schedule which re-insertion results in an unvisited neighbor: if $j \in \chi(\mathcal{S})$, then the corresponding neighbor $\mathcal{S}'_j$ of schedule $\mathcal{S}$ has not been considered yet.

Formally our implementation of the tabu search algorithm can be described as follows.

Algorithm '*tabu-search*'

1. Initialize an empty tabu list $\mathcal{T}$.
   Generate a given number of solutions by construction heuristics and create pool $\mathcal{Q}$.
   For each schedule $\mathcal{S} \in \mathcal{Q}$, find its critical path, initiate the list $\chi(\mathcal{S})$ of critical jobs for neighbor generation and use (12) to produce estimates σ_2 for all possible neighbors.
2. Repeat steps 3–4 while the computation time is less than the pre-specified time limit.
3. Find solution $\mathcal{S} \in \mathcal{Q}$ which neighbor $\mathcal{S}'_j$ has the smallest estimate σ_2,

$$\sigma_2\left(\mathcal{S}'_j\right) = \min_{\mathcal{S}\in\mathcal{Q}} \min_{k\in\chi(\mathcal{S})} \left\{\sigma_2\left(\mathcal{S}'_k\right) \mid \mathcal{S}'_k \text{ is a obtained from } \mathcal{S} \text{ by re-inserting } k\right\}. \tag{13}$$

   Generate $\mathcal{S}'_j$ using procedure 'Insert' and remove job j from list $\chi(\mathcal{S})$.
   Replace the record of the best solution found so far, if $\mathcal{S}'_j$ beats it.
   If $\mathcal{S}'_j$ is the only non-visited neighbor of $\mathcal{S}$ while all other neighbors have been visited, remove $\mathcal{S}$ from pool $\mathcal{Q}$.
4. If the critical path of $\mathcal{S}'_j$ belongs to tabu list $\mathcal{T}$, update the hit counter and the serial number of the corresponding entry in $\mathcal{T}$ and eliminate $\mathcal{S}'_j$ from further consideration.
   Otherwise perform Steps 4.1–4.2.
   4.1. Add the critical path of $\mathcal{S}'_j$ to $\mathcal{T}$ setting the hit counter to 1 and serial number to the current number of $\mathcal{S}'_j$.
      Calculate estimates of all possible neighbors of solution $\mathcal{S}'_j$.
      If the size of $\mathcal{T}$ reaches a given threshold value, remove half of the entries which have the smallest hit counter values; in case of ties eliminate the entries with the smallest serial numbers.
   4.2. Add $\mathcal{S}'_j$ to the pool of current solutions $\mathcal{Q}$. If the size of $\mathcal{Q}$ reaches a given threshold value, $\mathcal{Q}$ is emptied and refilled with a given number of schedules generated by construction heuristics.

In our implementation, we do not re-calculate estimates $\sigma_2\left(\mathcal{S}'_j\right)$ in Step 3 in each iteration; instead for every schedule $\mathcal{S}'_j$ in pool $\mathcal{Q}$ we keep the estimates of its neighbors.

Step 4 involves the search in tabu list $\mathcal{T}$ to verify if the critical path of the generated new solution $\mathcal{S}'_j$ belongs to it. In order to perform that search operation fast enough, we implement tabu list $\mathcal{T}$ as a radix tree (Patricia trie) with alphabet $\{1, 2, \ldots, 2n\}$ needed to specify critical operations and their sequences.

4. Construction heuristics

To the best of our knowledge, there are no heuristics developed specifically for the coupled-operation scheduling problem with exact time-lags. The most relevant algorithms presented in [22] are aimed at solving a more general version of the problem in which time-lags are flexible, i.e., can vary within given limits (see Section 2). Computational experiments show that for the problem with flexible time-lags randomized construction heuristic outperforms local-search algorithms. In this section we adapt the most successful construction heuristic JOIN-DECOMPOSE [22] to our problem and, in addition, we develop a simple but efficient EARLIEST-FIT dispatching rule. Both heuristics are used to evaluate the performance of tabu search in the computational experiments of Section 5.

JOIN-DECOMPOSE (JD) is a construction heuristic introduced in [22] for the problem with flexible time-lags. The algorithm first iteratively creates composite jobs at the *join* stage; then composite jobs are scheduled one after another and after that they are broken down into individual jobs at the *decompose* stage. The algorithm maintains a pool of composite jobs which is initiated as the set of individual input jobs. At the join stage, two jobs in the pool are selected and replaced by their combination: a new composite job. The procedure is repeated until the jobs in the pool cannot be combined any more. Composite jobs are placed into a schedule one after another without interleaving and at the decompose stage they are broken down into individual jobs.

In [22], the authors describe 12 rules for generating combined jobs and additional rules for selecting the best combined job. The same rules can be used for the problem with exact time-lags by setting the flexibility parameter to zero. In our experiments, we use the randomized version of heuristic JD with randomness introduced for rule selected. Repeated application of the algorithm generates many feasible schedules within a given time limit from which the best one is selected.

In addition to adopting JD from [22], we propose an alternative heuristic EARLIEST-FIT (EF). It is based on a greedy dispatching rule which schedules the jobs one at a time. At each iteration, it selects one unscheduled job and includes it into the current partial schedule at the earliest feasible starting time so that no job already scheduled is postponed and all time-lags are observed. Formally, for a coupled-operation job j, starting time t is feasible if the machine is idle in time intervals $\left[t, t + p_{a_j}\right]$ and $\left[t + p_{a_j} + L_j, t + p_{a_j} + L_j + p_{b_j}\right]$ where operations a_j and b_j can be processed. In our experiments, heuristic EF is run multiple times in a similar fashion as heuristic JD; randomness in this case is achieved through a probabilistic choice of a current job for inserting in the partial schedule.

Both algorithms JOIN-DECOMPOSE and EARLIEST-FIT have time complexity $O\left(n^2\right)$.

5. Computational experiments

In order to evaluate the performance of our tabu search algorithm, we compare it with the two construction heuristics JD and EF presented in Section 4. JD is a winning algorithm for the coupled-operation problem with flexible time-lags [22] and hence it can be considered as a benchmark. For this reason we mainly repeat the experimental design from [22] with some extensions.

Notice that the computational experiments in [22] are aimed at analyzing the algorithms for the coupled-operation problem with flexible time-lags which satisfy (2), and therefore for time-lag generation, the following two parameters are used:

- parameter α for generating the lower bound ℓ_j for a time-lag of job $j \in N$;
- parameter β for generating the value of the time-lag flexibility f_j,

$$f_j = u_j - \ell_j.$$

Since in the problem we study the time-lags are fixed with $f_j = 0,\ j \in N$, in the rest of this section, comparing our algorithms with those from [22], we consider instance generation strategy from [22] with the flexibility parameter β set equal to zero.

5.1. Instance generation

Computational experiments have been performed on two classes of instances: SIM and MIX. In the SIM class, all jobs have similar time-lags; in the MIX class, job time-lags differ significantly.

Following the strategy from [22], we define processing times p_{a_j} and p_{b_j} as integers drawn from a discrete uniform distribution,

$$p_{a_j}, p_{b_j} \in [1, 100] \quad \text{for each } j \in N.$$

The time-lags L_j are also integers sampled from a uniform distribution depending on parameter α (its choice will be explained later on) and on the average processing time

$$p_{\text{avg}} = \frac{1}{2n} \sum_{j \in N} \left(p_{a_j} + p_{b_j}\right).$$

For the SIM class the following formula is used:

- $L_j \in \left[0.9\alpha p_{\text{avg}}, 1.1\alpha p_{\text{avg}}\right]$

with $\alpha \in \left\{1, \frac{n}{5}, \frac{2n}{5}\right\}$ (as in [22]).

For the MIX class, we distinguish between two subclasses of instances, MIX_1 and MIX_2, depending on the time-lag calculation formula:

- $L_j \in [1, \alpha]$ is used for class MIX_1, where $\alpha \in \{5, 10, 25, 50\}$ (as in [22]);
- $L_j \in \left[1, \alpha p_{\text{avg}}\right]$ is used for class MIX_2, where $\alpha \in \left\{1, \frac{n}{5}, \frac{2n}{5}\right\}$.

Notice that, the MIX instances from [22] correspond to our subclass MIX_1. They are characterized by relatively small time-lags in comparison with the average processing time of 50. As a consequence, feasible solutions for such instances have only a few interleaving jobs and do not represent the complexity of the problem.

We keep subclass MIX_1 in our experiments for completeness of the analysis and to allow a one-to-one comparison with the results from [22]. In addition, we introduce subclass MIX_2 with relatively large time-lags which allow for complex interleaving of multiple jobs.

For each class SIM, MIX_1 and MIX_2, we consider all combinations of $n \in \{20, 30, 50, 100\}$ and α, and for each combination we generate 10 instances.

All algorithms were coded in C and run on one core of a PC with processor Intel Quad Core 2.5 GHz with 3 GB of RAM. The parameters of the tabu search algorithm were tuned after several preliminary experiments: we set 500 as the maximum length of the tabu list $\mathcal{T}$ and 200 as the maximum length of pool $\mathcal{Q}$. Each time the pool is emptied, 50 random solutions are generated using one of the construction heuristics from Section 4: JOIN-DECOMPOSE or EARLIEST-FIT. Depending on the heuristic used, we call the corresponding tabu search algorithm as TS-JD and TS-EF. We compare the results of the two tabu search algorithms with multiple runs of the two heuristics JD and EF applied independently.

5.2. Results

The performance of an algorithm on a given instance is measured in terms of the relative deviation ρ from the lower bound:

$$\rho = \frac{C_{\max}^{\text{Best}} - LB}{LB},$$

where $C_{\max}^{\text{Best}}$ is the smallest makespan found by the algorithm and LB is the lower bound. The overall performance of an algorithm on a set of instances is then measured as the average value $\overline{\rho}$ of the relative deviation over all instances considered. In the summary tables, we also indicate the time that the algorithm takes to generate best solutions, taken as the average value T_{avg} over all instances of the dataset.

For lower bound calculation we use the approach described in [22]. It starts with identifying a subset of jobs $N_1 \subseteq N$ containing the jobs which cannot be interleaved with another job from N. In particular, job j is included in this set if none of the sequences $\left(a_k, a_j, b_j, b_k\right)$, $\left(a_k, a_j, b_k, b_j\right)$, $\left(a_j, a_k, b_j, b_k\right)$ or $\left(a_j, a_k, b_k, b_j\right)$ is feasible for any $k \in N \setminus \{j\}$. Then the contribution of all jobs from N_1 is

$$LB\left(N_1\right) = \sum_{j \in N_1} \left(p_{a_j} + L_j + p_{b_j}\right).$$

The contribution of the remaining jobs $N_2 = N \setminus N_1$, which can be interleaved, is no less than their total processing requirement and it is also no less than the length of the longest job in that set:

$$LB\left(N_2\right) = \max\left\{\sum_{j \in N_2} \left(p_{a_j} + p_{b_j}\right), \max_{j \in N_2} \left\{p_{a_j} + L_j + p_{b_j}\right\}\right\}.$$

The lower bound is then

$$LB = LB\left(N_1\right) + LB\left(N_2\right).$$

In the experiments, we set up a time limit of 600 s for tabu search and for multiple runs of each of the two heuristics. Each time a new solution is generated, its makespan is compared with the lower bound LB to check if an optimum is found and the program may terminate immediately. This happened frequently only in our experiments with MIX_1 instances.

Table 2 summarizes the average accuracy of the two versions of the tabu search algorithm TS-EF and TS-JD and the two construction heuristics EF and JD on classes MIX and SIM. Algorithms are listed in the order of their overall performance measured in terms of the average value $\overline{\rho}$ of the relative ratio ρ calculated over 10 test instances. The second characteristic of each algorithm is the average time (in seconds) of finding the best solution.

Tabu search algorithms outperform construction heuristics in terms of the solution quality. JD construction heuristic tends to generate the best solution early and then it keeps running until the 600-second limit is reached without any improvement. Each version of tabu search continues improving the best solution for a longer period of time in comparison

Table 2
A summary of the results for all instances.

Algorithm	Overall		MIX_1		MIX_2		SIM	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.084	200	0.056	121	0.100	277	0.120	357
TS-JD	0.090	150	0.057	83	0.120	322	0.128	349
EF	0.198	186	0.071	115	0.250	272	0.336	281
JD	0.292	131	0.062	2	0.547	126	0.607	145

Table 3
Value of $\overline{\rho}$ for MIX instances obtained by the experiment in [22] with the parameter $\beta = 0$.

	Algorithm	MIX ($\beta = 0$)
Best randomized heuristics	RC	0.073
	C	0.077
Best local search	SMD	0.074
	RMD	0.086
	LHTS	0.093

Table 4
Value of $\overline{\rho}$ for SIM instances depending on the instance size.

Algorithm	SIM			
Input size n	20	30	50	100
TS-EF	0.099	0.112	0.116	0.151
TS-JD	0.101	0.114	0.119	0.178
EF	0.188	0.264	0.377	0.636
JD	0.240	0.287	0.483	1.419

Table 5
Value of $\overline{\rho}$ for MIX instances depending on the instance size.

Algorithm	MIX_1				MIX_2			
Input size n	20	30	50	100	20	30	50	100
TS-EF	0.050	0.059	0.057	0.060	0.076	0.089	0.094	0.140
TS-JD	0.051	0.059	0.059	0.060	0.077	0.090	0.095	0.218
EF	0.056	0.066	0.077	0.086	0.145	0.200	0.272	0.382
JD	0.055	0.066	0.064	0.064	0.175	0.255	0.470	1.287

with the corresponding construction heuristic. Comparing the behavior of the algorithms on MIX and SIM instances we observe that, as a rule, all algorithms find solutions closer to the lower bound for MIX_1 instances.

For comparison purposes, we re-produce in Table 3 the best results from [22] obtained for instances with fixed time-lags (flexibility parameter β is 0). Such instances are comparable to our class MIX_1. Heuristics C and RC are the counterparts of our heuristic JD with additional rules dealing with job flexibility parameter β at the JOIN stage. Heuristic RC generates multiple schedules applying selection rules randomly; heuristic C generates only one schedule using one selection rule. Heuristics SMD, RMD and LHTS are local search algorithms based on operation re-insertion neighborhood: given a permutation of operations, an operation from the critical path is removed and re-inserted in another random position. Differently from our neighborhood, operation re-insertion can result in infeasible permutations which are discarded. SMD and RMD are descent algorithms, which start the search with random solutions and seeded solutions, respectively; LHTS is a tabu search algorithm. It is easy to see that our tabu search algorithms TS-JD and TS-EF outperform all best algorithms from [22]. Notice also that local search algorithms SMD, RMD and LHTS are often unable to find better solutions than those found by construction heuristics RC and C.

Tables 4 and 5 illustrate the behavior of the algorithms on instances of different sizes. For both classes of instances, MIX and SIM, tabu search algorithms produce high quality solutions whichever the size of the instance is; the accuracy only slightly deteriorates for larger values of n. Construction heuristics perform similarly on MIX_1 instances, while they show a substantial deterioration of the solution quality when n increases for MIX_2 and SIM classes. Heuristic JD performs particularly poorly on instances with $n = 100$ jobs resulting in deviation values $\overline{\rho}$ being more than twice the deviation of EF heuristic. Notice that $\overline{\rho}$ values are quite similar for all instance sizes in MIX_1, which supports our observation about the special structure of MIX_1 instances: their feasible solutions have a relatively small number of interleaving jobs and their optimal grouping can be found early in the search.

Tables 6–8 illustrate the behavior of the algorithms on instances with different values of α-parameter (see Section 5.1). Observe that $\overline{\rho}$-value for construction heuristics increases as α increases. Differently, tabu search algorithms running on

Table 6
Value of $\overline{\rho}$ for SIM instances depending on the α-value.

Algorithm	SIM					
α	1		$\frac{n}{5}$		$\frac{2n}{5}$	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.183	295	0.082	388	0.094	389
TS-JD	0.188	260	0.085	400	0.110	388
EF	0.281	276	0.334	261	0.484	307
JD	0.212	64	0.533	185	1.076	188

Table 7
Value of $\overline{\rho}$ for MIX_1 instances depending on the α-value.

Algorithm	MIX_1							
α	5		10		25		50	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.006	60	0.028	59	0.081	128	0.111	273
TS-JD	0.006	15	0.028	17	0.083	130	0.113	172
JD	0.006	0	0.028	0	0.083	0	0.132	8
EF	0.006	2	0.028	59	0.089	128	0.162	273

Table 8
Value of $\overline{\rho}$ for MIX_2 instances depending on the α-value.

Algorithm	MIX_2					
α	1		$\frac{n}{5}$		$\frac{2n}{5}$	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.113	157	0.080	318	0.106	358
TS-JD	0.113	184	0.082	399	0.165	383
EF	0.161	185	0.277	341	0.310	292
JD	0.129	2	0.447	182	1.065	195

SIM and MIX_2 instances seem to perform better when $\alpha > 1$ in comparison with $\alpha = 1$. This suggests that our tabu search design is much more powerful than construction heuristics to generate schedules of good quality even if there are many interleaving jobs. For MIX_1 instances, all algorithms reach similar $\overline{\rho}$-values due to the special structure of the class.

Summarizing, computational experiments show that the proposed tabu search algorithms based on job re-insertion provide solutions of higher quality than the algorithms previously known. In particular, they compare favorably with construction heuristics EF and JD. Notice that JD is the best published algorithm for the coupled-operation problem with flexible time-lags.

JD performs poorly in comparison with our heuristic EF when run for a long period of time since it generates solutions of a similar structure. Differently, heuristic EF explores broadly the solution space generating a wider variety of solutions in comparison with JD. However, JD heuristic is an appropriate choice for instances of special structure, when good solutions are needed within a short period of time.

6. Conclusions

In this paper we propose a tabu search algorithm for scheduling n coupled-operation jobs with exact time-lags on a single machine. In order to explore possible solution representations, we study the special case of the problem with the fixed order of the first n operations and establish its NP-hardness even for the special case of unit-time operations. Due to this reason we select an alternative solution representation based on the permutation of all $2n$ operations.

For the neighbor generation strategy we adopt the insertion technique proposed in [13]. We show that an optimal insertion of a job in a partial schedule can be found in polynomial time. The algorithm uses the disjunctive graph model and exploits a so-called short cycle property. Notice that in the past the short cycle theory was successfully applied mainly to some special job shop problems, see [13–15].

The tabu search algorithm we develop differs from the traditional version by maintaining a pool of current solutions instead of a single solution. Due to this reason, it explores a broader part of the solution space. With a larger size of the neighborhood, it becomes particularly important to select potentially good neighbors fast enough. To achieve this we calculate makespan estimates for the neighbors and select the candidate with the smallest estimate.

The following two enhancements are introduced in tabu list implementation: (1) solutions are represented by their critical paths and (2) the entries of the tabu list are prioritized depending on how often the corresponding solutions are generated; whenever there is a need to reduce the size of the tabu list, the entries with the lowest frequency are removed first.

Computational experiments demonstrate that the proposed tabu search is effective for solving the coupled-operation problem with exact time-lags. It outperforms the earlier heuristics developed for the more general problem with flexible time-lags, in particular the join-and-decompose (JD) heuristic, which is considered to be the most successful among those studied in [22]. We believe that the main reason of good performance of our tabu search algorithm is related to the neighbor generation strategy which always produces a feasible solution, while neighbor generation strategy from [22] often results in infeasible solutions.

To summarize, the main outcomes of our study include the analysis of possible solution representations, an efficient neighbor generation strategy and two enhancements for tabu search implementation. It should be noted that the improvements achieved for the coupled-operation problem with exact time-lags cannot be immediately transferred to the more general problem with flexible time-lags. In particular, the insertion technique for neighbor generations is not applicable to the flexible time-lag version of the problem since the short cycle property is not satisfied for it.

Our work leads to several new research directions. First, it seems interesting to study other compact representations of feasible coupled-operation schedules. In particular, a schedule can be given by two sequences π_a and π_b of a- and b-operations. The associated problem is to merge two sequences π_a and π_b so that the resulting coupled-operation schedule is feasible and the makespan is minimum. If that problem can be solved in polynomial time, the new solution representation can serve the basis for a new set of local search algorithms with a reduced solution space.

Another interesting research direction is related to the neighbor generation strategy based on the job re-insertion problem. Proving that the proposed neighborhood is opt-connected would provide a stronger analytical argument to support the efficiency of our tabu search algorithm. Notice that opt-connectivity remains an open question for a similar neighborhood proposed in [14] for the blocking job-shop problem.

Acknowledgments

The authors are grateful to two anonymous referees for careful reading of the text and suggested improvements.

References

[1] A. Ageev, A. Barburin, Approximation algorithms for UET scheduling problems with exact delays, Operations Research Letters 35 (4) (2007) 533–540.
[2] A. Ageev, A. Kononov, Approximation algorithms for scheduling problems with exact delays, Lecture Notes in Computer Science 4368 (2007) 1–14.
[3] D. Ahr, J. Békési, G. Galambos, M. Oswald, G. Reinelt, An exact algorithm for scheduling identical coupled tasks, Mathematical Methods of Operations Research 59 (2) (2004) 193–203.
[4] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.
[5] P. Baptiste, A note on scheduling identical coupled tasks in logarithmic time, Discrete Applied Mathematics 158 (5) (2010) 583–587.
[6] J. Blazewicz, K. Ecker, T. Kis, C.N. Potts, M. Tanas, J. Whitehead, Scheduling of coupled tasks with unit processing times, Journal of Scheduling 13 (5) (2010) 453–461.
[7] P. Brucker, Scheduling Algorithms, Springer, 2007.
[8] P. Brucker, S. Knust, C. Oguz, Scheduling chains with identical jobs and constant delays on a single machine, Mathematical Methods of Operations Research 63 (1) (2006).
[9] A. Condotta, N.V. Shakhlevich, Scheduling patient appointments via multi-level template: a case study in chemotherapy (submitted for publication).
[10] M. Elshafei, H.D. Sherali, J.C. Smith, Radar pulse interleaving for multi-target tracking, Naval Research Logistics 51 (1) (2004) 72–94.
[11] A. Farina, P. Neri, Multitarget interleaved tracking for phased-array radar, IEE Proceedings F (Communications, Radar and Signal Processing) 127 (4) (1980) 312–318.
[12] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 4 (1979) 287–326.
[13] H. Gröflin, A. Klinkert, Feasible insertions in job shop scheduling, short cycles and stable sets, European Journal of Operational Research 177 (2) (2007) 763–785.
[14] H. Gröflin, A. Klinkert, A new neighborhood and tabu search for the blocking job shop, Discrete Applied Mathematics 157 (17) (2009) 3643–3655.
[15] H. Gröflin, A. Klinkert, N. Dinh, Feasible job insertions in the multi-processor-task job shop, European Journal of Operational Research 185 (3) (2008) 1308–1318.
[16] A. Hertz, E. Taillard, D. de Werra, Tabu search, in: J.K. Lenstra, E. Aarts (Eds.), Local Search in Combinatorial Optimization, Princeton University Press, 2003, pp. 121–136.
[17] J. Hurink, J. Keuchel, Local search algorithms for a single-machine scheduling problem with positive and negative time-lags, Discrete Applied Mathematics 112 (1–3) (2001) 179–197.
[18] J.Y.T. Leung, H.R. Zhao, Minimizing sum of completion times and makespan in master–slave systems, IEEE Transactions on Computers 55 (8) (2006).
[19] C.K.Y. Lin, K.B. Haley, Scheduling two-phase jobs with arbitrary time lags in a single-server system, IMA Journal of Mathematics Applied in Business and Industry 5 (1993) 143–161.
[20] A. Munier, F. Sourd, Scheduling chains on a single machine with non-negative time lags, Mathematical Methods of Operations Research 57 (1) (2003) 111–123.
[21] A.J. Orman, C.N. Potts, On the complexity of coupled-task scheduling, Discrete Applied Mathematics 72 (1–2) (1997) 141–154.
[22] C.N. Potts, J.D. Whitehead, Heuristics for a coupled-operation scheduling problem, Journal of the Operational Research Society 58 (10) (2007) 1375–1388.
[23] R.D. Shapiro, Scheduling coupled tasks, Naval Research Logistics 27 (3) (1980) 489–498.
[24] E. Wikum, D.C. Llewellyn, G.L. Nemhauser, One-machine generalized precedence constrained scheduling problems, Operations Research Letters 16 (2) (1994) 87–99.
[25] E. Winter, P. Baptiste, On scheduling a multifunction radar, Aerospace Science and Technology 11 (4) (2007) 289–294.
[26] W. Yu, H. Hoogeveen, J.K. Lenstra, Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard, Journal of Scheduling 7 (5) (2004) 333–348.