RAIRO-Oper. Res. 46 (2012) 335–353
DOI: 10.1051/ro/2012020

RAIRO Operations Research
www.rairo-ro.org

NEW ALGORITHMS FOR COUPLED TASKS SCHEDULING – A SURVEY

Jacek Blazewicz[1,2], Grzegorz Pawlak[2], Michal Tanas[3] and Wojciech Wojciechowicz[2]

Abstract. The coupled tasks scheduling problem is a class of scheduling problems introduced for beam steering software of sophisticated radar devices, called phased arrays. Due to increasing popularity of such radars, the importance of coupled tasks scheduling is constantly growing. Unfortunately, most of the coupled tasks problems are NP-hard, and only a few practically usable algorithms for such problems were found. This paper provides a survey of already known complexity results of various variants of coupled tasks problems. Then, it complements previous results by providing experimental results of two new polynomial algorithms for coupled tasks scheduling, which are: an exact algorithm for $1|(1,4,1), \textit{strict chains}|C_{\max}$ problem, and a fast heuristic algorithm for more general $1|(1,2k,1), \textit{strict chains}|C_{\max}$ problem, where $k \in \mathbb{N}$.

Keywords. Coupled tasks, scheduling, complexity theory, heuristic algorithms.

Mathematics Subject Classification. 9002, 9008, 90B30, 90B35, 90C27, 90C59, 90C60.

Received October 5, 2012. Accepted October 12, 2012.

[1] Institute of Bioorganic Chemistry, Polish Academy of Sciences, ul. Z. Noskowskiego 12/14, 61-704 Poznan, Poland. jblazewicz@cs.put.poznan.pl

[2] Institute of Computing Science, Poznan University of Technology, ul. Piotrowo 2, 60-965 Poznan, Poland

[3] Computer Science Division, Physics Faculty, Adam Mickiewicz University, ul. Umultowska 85, 61-614 Poznan, Poland

Article published by EDP Sciences

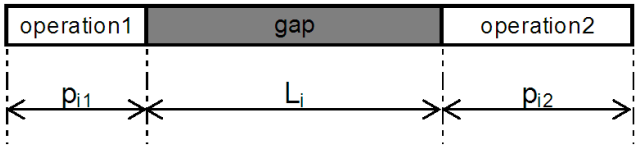

FIGURE 1. An example of a coupled task.

1. INTRODUCTION

According to the definition given by Baker in [3], scheduling is a problem of allocating resources over time in order to perform a given set of tasks. In practice, resources are processors, tools, manpower or money. Tasks could be characterized by parameters like due dates, deadlines, ready times, precedence constraints etc. The schedule quality can be measured using various criteria, depending *e.g.* on the optimization function. A survey of the many important results can be found in the handbooks [10, 25]. The commonly used notation, as well as general formulation of scheduling problems are provided by Brucker [13] and Blazewicz *et al.* [10].

The coupled tasks problem was firstly stated in early eighties by Shapiro [31]. A task is called *coupled* if it contains two operations T_{i1} and T_{i2} where processing of the second operation can start only after given delay (called *gap*) succeeding completion of the first operation. An example of a coupled task is given in Figure 1

This problem was used to model phased array radars, which were being installed on board of US Navy ships to track incoming guided missiles. According to the classic definition (see [32]) a phased array radar is "a directive antenna made up of individual radiating antennas, or elements, which generate a radiation pattern whose shape and direction is determined by the relative phases and amplitudes of the currents at the individual elements". For the scope of this paper, we wish to emphasize the most distinctive feature of phased array radar - its ability to perform beam steering and directing in pure electronic way, without any mechanically rotating antennas. This gives the phased array radar unique ability to transmit several pulses into different directions before the echo of the first pulse manages to return to the antenna. Therefore, a single phased array radar is able to track several targets simultaneously, but only if the antenna is directed to collect echoes in the perfectly appropriate time units. This is the classical coupled tasks example: the first operation is the transmission of a pulse, the gap is the period where the pulse travels somewhere in the air but the radar itself is idle and the second operation is the reception of the echo.

From this time importance of such type of scheduling constantly grows, due to increasing popularity of phased array radars in various military applications (see for example Orman *et al.* [29]). Moreover, due to replacement of the first generation phased array radars by various navies and the availability of obsolete but still operational military radars to some civilian institutions, the last years bring

several new applications of phased array radars and applications of coupled tasks scheduling. The most notable application is an experimental phased array weather radar capable of measuring the speed of wind inside violent weather phenomena, like hurricanes and tornadoes (see [22]). Some new applications of coupled tasks scheduling were described in [21].

Brauner, Finke, Lehoux-Lebacque, Potts and Whitehead [12] used the coupled tasks scheduling approach for modeling a single machine no-wait robotic cell in production systems. In general, a robotic cell is a fully automated production node in a flexible manufacturing system (FMS). Such systems constantly become more and more popular in industrial plants due to their flexibility. The important feature of the robotic cell is that its input and output are transported by a robot and the transport time is greater than zero(*cf.* [11]). Therefore, the periods where the robot carries parts are modeled by operations and the period in which the robot is idle and waits for the output to be processed, is modeled by the gap.

For the purpose of modeling various applications described above, a large number of different subproblems and variants of coupled tasks scheduling were formulated and investigated in the past.

Taking into consideration applications, the case of scheduling identical coupled tasks with unit processing times on a single machine is specially interesting, *i.e.* the problem $1|(1, L, 1)|C_{\max}$ and its various subproblems (*cf.* Sect. 2 for problem notation). The strong NP-completeness of the case, where the precedence constraints graph is a general graph, was shown in [8]. On the other hand, polynomial solvability of the $1|(1, 2, 1), \textit{strict tree}|C_{\max}$ case was shown in [34] and polynomial solvability of the $1|(1, L = 2k, 1), \textit{strict chains}|C_{\max}$ was shown in [15]. These results were generalized by a proof of polynomial solvability of the $1|(1, L, 1), \textit{strict chains}|C_{\max}$ problem in[16]. However, the algorithms used for the proofs of polynomial solvability are not well suited for practical applications due to the huge complexity constants. Therefore, in this paper, we complement the above results by presenting analysis of practically usable algorithms (exact and a heuristic) for the problem $1|(1, 4, 1), \textit{strict chains}|C_{\max}$ and thus present a way to solve the considered problem in practice.

In general, the problem analyzed in the paper concerns coupled tasks scheduling on a single processor. After a detailed introduction of the subject and a presentation of the most interesting results in the area, new scheduling algorithms, complementing the existing ones will be presented. The organization of the paper is as follows. Section 2 contains problem formulation while the state of the art of the most important results in the area of coupled tasks scheduling is given in Section 3. The algorithms tested in our experiment are described in Section 4. The results of the experiment are presented in Section 5. We conclude in Section 6.

2. General problem formulation

In general, a scheduling problem is characterized by two sets: the set $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ of *tasks* to be processed and the set $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$

of *processors (machines)* being able to process the tasks. Thus, scheduling means finding an assignment of processors from $\mathcal{P}$ to tasks from $\mathcal{T}$ in order to complete all these tasks. Sometimes a single task T_j may be divided into a sequence of smaller subtasks, called *operations*.

Our scheduling problem has certain specificity which will be explained in the following section. As defined in Section 1 in the coupled task scheduling problem, each task T_i is composed of two operations T_{i1} and T_{i2} which must be processed one after the other on the same processor at the distance (gap) of L units. Their *processing times* are p_{i1} and p_{i2}, respectively. Tasks may be also ordered by the *precedence relation*. Further details are given below together with a description of the notation of scheduling problems.

To describe scheduling problems the standard $\alpha|\beta|\gamma$ notation, introduced by Graham, Lawler, Lenstra and Rinnooy Kan [19] and then further extended by Blazewicz *et al.* [10], is commonly used.

Field α characterizes the processors in the system. In this paper two types of processor systems are considered:

- Single machine system. In such case $\alpha = 1$.
- Systems which contain fixed number of parallel processors. In such case $\alpha = Pm$ where m is the number of processors in the system. (For the description of other types of processors see [10] or [6,7]).

Field β characterizes the set of tasks and additional resources. In this paper the following notation in this field is used:

- $(p_{j1}, [L_j^{\min}, L_j^{\max}], p_{j2})$ means that the tasks are coupled, where
  - p_{j1} is the processing time of the first operation.
  - $[L_j^{\min}, L_j^{\max}]$ is the lower and upper bound of the gap. If $\forall_j\ L_j^{\min} = L_j^{\max}$ this field is usually shortened to L_j and such problem is called the *exact gap* problem.
  - p_{j2} is the processing time of the second operation.
- $p_j = 1$ means that the processing time of each task is equal to 1. This notation is used for non-coupled task problems.
- *strict* signifies that the precedence constraints are strict. It implies that the processing of the first operation of a subsequent task is not started until the processing of the second operation of the preceding task is finished (*i.e.* strict precedence constraints mean that $T_i \to T_j \Longrightarrow T_{i2} \to T_{j1}$). The other type of precedence constraints are *weak* precedence constraints where $T_i \to T_j \Longrightarrow T_{i1} \to T_{j1} \wedge T_{i2} \to T_{j2}$. In general, precedence constraints may be represented by different graphs. In this paper, *chains*, *intrees*, *outtrees* and general graphs (*prec*) are used. An interested reader may consult [10] for the definitions of these graphs.
- r_j means different ready (arrival) times of tasks.
- *pmtn* means that the tasks are preemptive, *i.e.* processing of each tasks may be stopped at any moment and resumed later. Lack of *pmtn* in the description of the problem means that once started, processing of a task cannot be preempted until its completion.

Field γ characterizes the *optimality criterion* of the problem. In this paper the following criteria are considered:

- $C_{\max}$ is called *makespan* or *schedule length*. $C_{\max}$ is equal to completion time of the last processed task, *i.e.* $C_{\max} = \max_{j=1,\ldots,n}(C_j)$ where C_j is the completion time of the last operation of task T_j.
- $\sum C_j$ - *mean flow time* is the sum of completion times of all tasks in the instance, *i.e.* $\sum C_j = \sum_{j=1}^{n} C_j$, where C_j is the completion time of the last operation of task T_j.

A brief summary of the notation used in this paper is as follows:

- n is the number of coupled tasks,
- h is the number of chains in a precedence graph,
- L is the length of the gap,
- h_i is the number of coupled tasks in i-th chain,
- I is a given instance of the problem,
- S is a schedule, either complete or partial,
- *segment* is a coupled tasks partial schedule.

As an example, one of the problems considered in this paper $1|(1,4,1), \textit{strict chains}|C_{\max}$ shall be defined as follows:

- there is a single processor in the system,
- all tasks are coupled,
- all tasks are identical,
- all gaps are exact,
- processing time of each operation is equal to 1,
- length of each gap is equal to 4,
- precedence constrains are strict,
- precedence constraints' graph has a form of chains,
- the optimization criterion is $C_{\max}$.

We refer the reader to [8,10,13] for a more detailed description of the notation used in scheduling area.

3. Coupled tasks - the state of the art

For the purpose of modeling different applications described above, a large number of different subproblems and variants of coupled tasks scheduling were formulated and investigated in the past. This section tries to provide a brief survey of current coupled tasks research domain and to point out the most important results achieved.

3.1. Arbitrary processing times and exact gap

The coupled task problems with arbitrary processing times and exact gap were studied first by Orman and Potts [27]. Most of these problems were proved to be NP-hard in the strong sense, however, a few important polynomial algorithms were also found. It is interesting that the case of identical tasks, which seems to be the simplest, was solved by Ahr *et al.* [2] almost a decade after the other variants. The summary of the arbitrary processing times with exact gap case is as follows:

- NP-hard problems (in the strong sense)
  - $1|(a_j, L_j, b_j)|C_{\max}$ – Orman and Potts [27].
  - $1|(p_j, p_j, p_j)|C_{\max}$ – Orman and Potts [27].
  - $1|(a, L_j, b)|C_{\max}$ – Orman and Potts [27].
  - $1|(a, L, b_j)|C_{\max}$ – Orman and Potts [27].
- Polynomially solvable problems
  - $1|(a, L, b)|C_{\max}$ – Ahr, Bekesi, Galambos, Oswald and Reinelt [2] and a logarithmic complexity algorithm by Baptiste [4].
  - $1|(p, p, b_j)|C_{\max}$ – Orman and Potts [27].
  - $1|(p, L, p)|C_{\max}$ – Orman and Potts [27].

3.2. Unit processing times and exact gap

The coupled task problem with unit processing times and exact gap was introduced as a relaxation of the original problem, due to NP-hardness of the most interesting arbitrary processing times cases. The unit processing times variant of the coupled task problem was initially developed by Wenci Yu [35, 36] and then investigated by Blazewicz, Ecker and Tanas mainly in [8, 9, 15, 16]. The summary of the unit processing times with exact gap case is as follows:

- NP-hard problems (in the strong sense)
  - $1|(1, L_{ij}, 1), \textit{strict chains}|C_{\max}$ – Wenci Yu [35, 36].
  - $1|(1, L, 1), \textit{strict prec}|C_{\max}$ – Blazewicz, Ecker and Tanas [8, 9].
- Polynomially solvable problems
  - $1|(1, L, 1), \textit{strict chains}|C_{\max}$ – Ecker and Tanas proved polynomial solvability in [16], although with huge constant. A much more practically usable $O(n \log n)$ approximation is given in this paper (the FALE algorithm).
  - $1|(1, 2, 1), \textit{strict tree}|C_{\max}$ – Whitehead [34].
  - $1|(1, 3, 1), \textit{strict chains}|C_{\max}$ – Ecker and Tanas [15].
  - $1|(1, 4, 1), \textit{strict chains}|C_{\max}$ – the algorithm FEL4 in this paper.

3.3. Non-exact gap

As we defined, the variant of coupled tasks problem where the length of the gap is defined by an interval or a function, is called the *non-exact gap problem*. The non-exact gap problems were initially investigated by Gupta [20] and then

by Wenci Yu [35]. It is not surprising that even the simplest non-exact gap problems are NP-hard:

- NP-hard problems
  - $1|(a_j, [L_j, \infty], b_j)|C_{\max}$ – Gupta [20].
  - $1|(a_j, [L_j, L_j + \varepsilon], b_j)|C_{\max}$ – exact gap with tolerance (*i.e.* the gap may be slightly enlarged without infeasibility of solution) – Potts and Whitehead [30].

3.4. Problems with ready times

For every scheduling problem its variant with different ready times of the tasks may be formulated. In this case, a task cannot be started before a particular time moment, which is called ready time of the task. The coupled tasks problems with different ready times were analyzed by Brucker and Knust [14], and the results achieved were as follows:

- NP-hard problems
  - $1|(p_j, L, p_j), strict\ in-tree, r_j|\sum C_j$ – Brucker and Knust [14].
- Polynomially solvable problems
  - $1|(p_j, L, p_j), strict\ out-tree, r_j|\sum C_j$ – Brucker and Knust [14].
  - $1|(1, 1, 1), strict\ prec, r_j|\sum C_j$ – Brucker and Knust [14].

3.5. Cyclic problems

A cyclic case of a scheduling problem is when a set of tasks shall be repeated infinitely with a given frequency. Therefore, the goal of scheduling is not to minimize schedule length (which is in such case infinite by assumption) but to ensure that all tasks fit in the given time-window. Such a variant of scheduling problems is widely used in embedded machine controller, *e.g.* embedded computers which control machinery parameters. The cyclic case was investigated by Lehoux-Lebacque, Brauner and Finke [24], and the following result was achieved:

- Polynomially solvable problems
  - $1|(p_j, L, p_j), cycl|Ct_{\min}$ single machine cyclic coupled tasks with the minimization of the cycle time – Lehoux-Lebacque, Brauner and Finke [24].

3.6. Heuristic solutions for some coupled tasks scheduling problems

NP-hardness of important variants of coupled task scheduling problems caused the need for heuristic algorithms being able to generate acceptable solutions in practical applications. The most notable results were obtained in [1, 5, 17, 18, 23, 28, 30]. Table 1 summarizes the current state of the art of coupled tasks scheduling domain. To let this table remain simple, some special cases (*i.e.* cyclic problems) were omitted.

TABLE 1. Results.

Problem	Complexity	Best polynomial algorithm
$1\|(a_j, L_j, b_j)\|C_{\max}$	strongly NP-hard [27]	n/a unless P=NP
$1\|(p_j, p_j, p_j)\|C_{\max}$	strongly NP-hard [27]	n/a unless P=NP
$1\|(a, L_j, b)\|C_{\max}$	strongly NP-hard [27]	n/a unless P=NP
$1\|(a, L, b_j)\|C_{\max}$	strongly NP-hard [27]	n/a unless P=NP
$1\|(a, L, b)\|C_{\max}$	polynomial [2]	$O(\log n)$ [4]
$1\|(a_j, p, p)\|C_{\max}$	polynomial [27]	$O(n)$ [27]
$1\|(p, p, b_j)\|C_{\max}$	polynomial [27]	$O(n)$ [27]
$1\|(p, L, p)\|C_{\max}$	polynomial [27]	$O(n)$ [27]
$1\|(1, L, 1), \textit{strict chains}\|C_{\max}$	polynomial if $L = \text{const}$. [16]	$O(n \log n)$, and this paper
$1\|(1, 2, 1), \textit{strict tree}\|C_{\max}$	polynomial [34]	$O(n^2)$ [34]
$1\|(1, 3, 1), \textit{strict chains}\|C_{\max}$	polynomial [15]	$O(n \log n)$ [15]
$1\|(1, 4, 1), \textit{strict chains}\|C_{\max}$	polynomial - this paper	$O(n \log n)$ - FEL4 in this paper
$1\|(1, L_{ij}, 1), \textit{strict}]\textit{chains}\|C_{\max}$	NP-hard [36]	n/a unless P=NP
$1\|(1, L, 1), \textit{strict prec}\|C_{\max}$	strongly NP-hard [8,9]	n/a unless P=NP
$1\|(a_j, [L_j, \infty], b_j)\|C_{\max}$	NP-hard [20]	n/a unless P=NP
$1\|(a_j, [L_j, L_j + \varepsilon], b_j)\|C_{\max}$	NP-hard [30]	n/a unless P=NP
$1\|(p_j, L, p_j), \textit{strict in} - \textit{tree}, r_j\|\sum C_j$	NP-hard [14]	n/a unless P=NP
$1\|(p_j, L, p_j), \textit{strict out} - \textit{tree}, r_j\|\sum C_j$	polynomial [14]	$O(n^2)$ [14]
$1\|(1, 1, 1), \textit{strict prec}, r_j\|\sum C_j$	polynomial [14]	$O(n^2)$ [14]

4. POLYNOMIAL ALGORITHMS FOR $1|(1, 4, 1), \textit{strict chains}|C_{\max}$

As we mentioned in Section 3, we complement the existing results by presenting an analysis of practically usable algorithms (exact and a heuristic one) for $1|(1, 4, 1), \textit{strict chains}|C_{\max}$. Three algorithms will be given and analyzed. The first one is a greedy algorithm that is very fast but not necessarily optimal. The second - FEL4 - is an optimal algorithm specialized for the case where $L = 4$. The third one - FALE - is an approximation algorithm for any even gap length. In Section 5 they will be submitted to tests.

4.1. GREEDY ALGORITHM

The first algorithm considered is a simple and quick greedy algorithm that is used as the upper bound on the quality of the obtained solutions (*i.e.* no practically

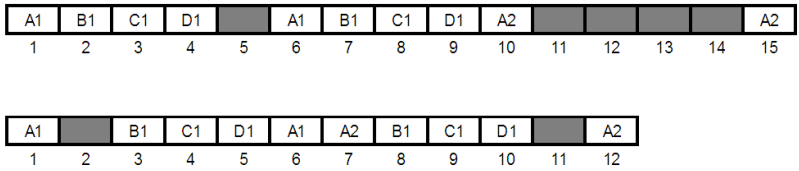

FIGURE 2. An example - greedy schedule (top) and the optimal one (bottom).

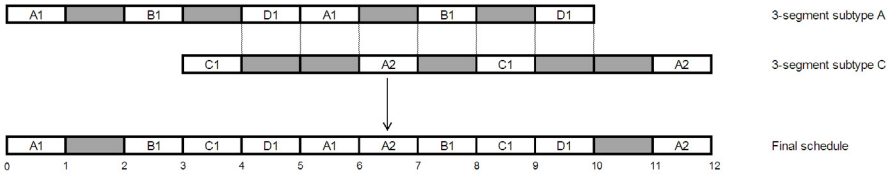

FIGURE 3. Constructing schedule from segments.

usable algorithm shall generate results worse than the greedy algorithm). The logic behind the greedy algorithm is as follows: it assigns to the first allowed position, the first available task (starting from the longest chain to the shortest one), and repeats this step until all tasks are scheduled. The complexity of the algorithm is obviously $O(n \log n)$ but it does not yield optimal solutions in some cases, as illustrated in the following example.

Example 4.1. Let us consider an example with four chains; the first one consists of 2 tasks while the remaining have only one task each. The makespan for greedy schedule is 15, while the optimal solution's makespan is 12. Both schedules are presented in Figure 2; idle times are marked with grey.

4.2. The FEL4 algorithm

The second algorithm analyzed in this paper is FEL4, which is a specialized exact algorithm designed for the $L = 4$ case. The algorithm is an improved and optimized version of a more general algorithm mentioned in [15] which was adopted to solve problem $1|(1, 4, 1), \textit{strict chains}|C_{\max}$. FEL4 is based on the idea that every feasible schedule can be presented as linearly ordered set of partial schedules (segments), each of which contains a limited number of tasks. An upper limit on this number does not depend on the instance size. In case of Example 1, these segments will have the form presented in Figure 3. It is obvious that there must be at least one task in a segment and that a segment should be constructed in $O(1)$ time because the size of the segment is limited by a constant.

Such approach divides our problem into maximum n subproblems, each of complexity $O(1)$. Since FEL4 guarantees the optimal results [15], it provides the lower bound on the makespan used for evaluation of other algorithms.

In the FEL4 algorithm, the following procedure is applied to find the optimal schedule. Firstly, the optimal order of the set of tasks shall be obtained:

1. Sort all chains in not ascending order of their lengths.
2. Convert the given instance I_1 of the problem $1|(1,4,1), \textit{strict chains}|C_{\max}$ to the instance I_2 of the problem $P5|pmtn|C_{\max}$ as follows:
   - The number of tasks in I_2 is equal to the number of chains in I_1.
   - Processing time of a task in I_2 is equal to the length of the chain corresponding to this task in I_1.
   - Preemptions are allowed only in integer points of time.
3. Compute the value of
$$D = \max\left\{\left\lceil \frac{1}{5}n \right\rceil, \max_{1\le i\le h} h_i\right\}$$
4. Create a schedule S_2 for the instance I_2 using the McNaughton's, [10,26] algorithm in time-window $\langle 0, D\rangle$.
5. Sort the coupled tasks in non-ascending order of starting times of their corresponding tasks in the schedule S_2. If for two tasks their starting times are equal, the task with the lower number processed on the machine, goes first. Create the list $\mathbb{L}$ containing all the coupled tasks in that order.

Secondly, when the coupled tasks are sorted in proper order, the optimal schedule shall be created:

6. Create FALE schedule S^{FALE} (see Sect. 4.3) for the instance I_1. Then compute the length $C_{\max}^{\text{FALE}}$ of the schedule S^{FALE} and the number and the number $IDLE(S^{\text{FALE}})$ of the units of idle time in the schedule S^{FALE}.
7. Create all possible segments S_i^1 which contain no less than 1 and no more than 5 tasks taken from the beginning of the list $\mathbb{L}$ and such that no segment is longer than 24 units of time. Let us denote by $\mathbb{S}$ the set containing all of those schedules.
8. $\mathbb{S}_2 = \emptyset$.
9. For each partial schedule $S_i \in \mathbb{S}$
   (a) Create the list $\mathbb{L}_i$ which is the list $\mathbb{L}$ with all tasks belonging to schedule S_i removed.
   (b) Create all possible segments S_j^2 which contain no less than 1 and no more than 5 tasks taken from the beginning of the list $\mathbb{L}_i$ and such that no segment is longer than 24 units of time.
   (c) For each segment S_j^1 create all possible schedules $S_{i,j}^2$ by concatenating S_i and S_j^2 and then setting the starting time of the first task of S_j^2 to $0, -1, -2, \ldots, -24$ relative to the end of the schedule S_i and then setting starting times of all other tasks in S_j^2 appropriately.

(d) For each schedule $S^2_{i,j}$ compute and store the following values:
   i. The length of the schedule $C_{\max}^{S^2_{i,j}}$.
   ii. The number $IDLE_{-24}(S^2_{i,j})$ of units of idle time appears in the schedule $S^2_{i,j}$ earlier than $C_{\max}^{S^2_{i,j}} - 24$ (*i.e.* such units of idle time that cannot be filled in further steps of the algorithm).

(e) For each schedule $S^2_{i,j}$ add this schedule to the set $\mathbb{S}_2$ if and only if all the following conditions are fulfilled:
   i. The schedule $S^2_{i,j}$ is feasible.
   ii. $C_{\max}^{S^2_{i,j}} - 24 \leq C_{\max}^{\mathrm{FALE}}$.
   iii. $IDLE_{-24}(S^2_{i,j}) \leq IDLE(S^{\mathrm{FALE}})$.

If any of the above condition is not fulfilled drop the schedule $S^2_{i,j}$.

10. For each pair of schedules $\left(S^2_{i,j}, S^2_{p,q}\right) \in \mathbb{S}_2 \times \mathbb{S}_2$ such that the last 24 units of time of both schedules are identical (the order in which are processed on the same machine in the corresponding parallel schedule, is not taken into account), remove the longer schedule from the set $\mathbb{S}_2$.
11. If all schedules in $\mathbb{S}_2$ contain exactly n tasks choose the shortest one and stop the algorithm.
12. $\mathbb{S} = \mathbb{S}_2$, go to 7.

The complexity of the FEL4 algorithm can be computed in the following way:

1. Sorting in the Step 1 has complexity $O(n \log n)$.
2. Conversion in the Step 2 has complexity $O(n)$ due to the obvious fact that $h \leq n$.
3. Computation of the formula in the Step 3 has the complexity $O(n)$ again because $h \leq n$.
4. The McNaughton algorithm in the Step 4 has the complexity $O(n)$ again because $h \leq n$.
5. Sorting in the Step 5 has complexity $O(n \log n)$.
6. The FALE algorithm in the Step 6 has complexity $O(n \log n)$ (see Sect. 4.3).
7. The number of segments created in the Step 7 does not depend on n and thus complexity of this step is $O(1)$.
8. The cardinality of the set $\mathbb{S}$ and so the maximum number of repetitions of the loop beginning in Step 9 does not depend on n and thus is $O(1)$.
9. Creation of the list in the Step 9a has complexity $O(n)$ because no schedule can contain more than n tasks.
10. Complexity of the Step 9b is the same as complexity of the Step 7 and thus $O(1)$.
11. The number of partial schedules in the Step 9d does not depend on n. Each subsequent computations has complexity $O(n)$ because any partial schedule may be longer than $6n$ units of time.
12. The number of partial schedules in the Step 9e does not depend on n. The feasibility check has complexity $O(1)$ because, for the chains precedence graph, each task may have no more than one precedessor.

13. The maximum number of pairs in the Step 10 does not depend on n because cardinality of none of the sets depends on n.
14. The maximum number of partial schedules in the Step 11 does not depend on n and thus complexity of this step is $O(1)$.

Therefore, the final complexity of the $FEL4$ algorithm is $O(n \log n)$. The more detailed complexity analysis and the proof of optimality of the general (*i.e.* for arbitrary L) version of the FEL4 algorithm is presented in [15]. The optimization done for $L = 4$ does not change the theoretical complexity of algorithm FEL4, but due to significant reduction of complexity constants makes the practical implementation of the algorithm possible.

4.3. The FALE algorithm

The third algorithm is called FALE (*cf.* [33]), which is an approximation algorithm, that works for any $L = 2k$. This algorithm consists of the following steps:

1. Sort all chains in not ascending order of their lengths.
2. Convert the instance I of the problem $1|(1, L, 1), \textit{strict chains}|C_{\max}$ to the instance I_2 of the problem $P(L+1)|pmtn|C_{\max}$ as follows:
   - The number of tasks in I_2 is equal to the number of chains in I
   - Processing time of a task in I_2 is equal to the length of the chain corresponding to this task in I.
   - Preemptions are allowed only in integer points of time.
3. Compute the value of

$$D = \max\left\{\left\lceil \frac{L}{\frac{L}{2}+1} \cdot \sum_{i=1}^{c} c_i \right\rceil, \max_{1\le j\le c} c_j\right\}.$$

4. Create a schedule S_2 for the instance I_2 using the McNaughton's algorithm in time-window $\langle 0, D\rangle$.
5. Sort the coupled tasks in not ascending order of starting times of their corresponding tasks in the schedule S_2. If for two tasks their starting times are equal, the task with the lower number processed on the machine, goes first. Create the list $\mathbb{L}$ containing all the coupled tasks in that order.
6. For each coupled tasks T_i assign its starting time s_i:

$$s_i = 2(m_i - 1) + s_i^{\text{pmtn}} + (L + 2).$$

This algorithm is capable of finding a near optimal solution, at the worst case L units of time longer than the optimal one(as there may be $\frac{1}{2}L$ unnecessary units of idle time at the beginning of the schedule and additional $\frac{1}{2}L$ unnecessary units of idle time at the end of the schedule). Since this value does not depend on the number of tasks, the FALE makespan goes to the optimality if the size of the instance goes to infinity.

The complexity of the FALE algorithm may be computed in a similar way as the one presented for the FEL4 algorithm (see Sect. 4.2). It is $O(n \log n)$.

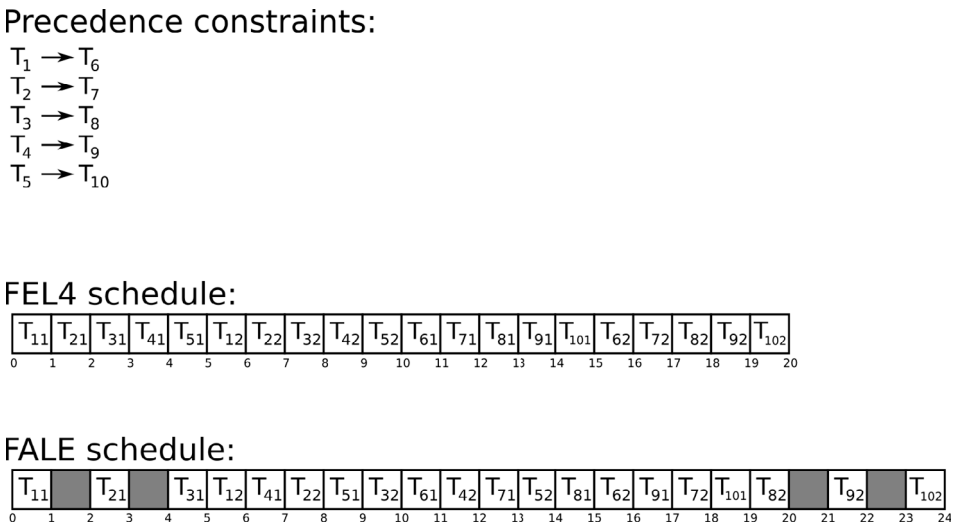

FIGURE 4. An example of schedules generated by FEL4 and FALE, respectively for the same instance of the problem. At the same time the example illustrates the worst case of the $FALE$ algorithm, where the $FALE$ schedule is L units of time longer than the optimal one.

Example 4.2. To visualize a difference between FEL4 and FALE algorithms let us consider a simple instance which contains 10 tasks in 5 chains. The schedules generated by FEL4 and FALE algorithms for such instance are presented in the Figure 4.

5. Computational experiment

5.1. Testbed

In this section, the computational results are presented. The analysis focused on two aspects - the computational time needed to construct a schedule, and a schedule quality measured as the deviation of achieved solution makespan and the optimal one. All algorithms were tested using sets of tasks with varied sizes (from 2 up to 838 tasks). For each instance size (thus the number of coupled tasks it contains), randomly generated instances were prepared:

- for instances of sizes between 2 and 20 tasks, the number of generated instances were equal to the size of instance,
- for instances larger than 20 tasks, 20 instances were generated.

In total, 16 569 testing sets were prepared. The instances were generated randomly, with the limit on the maximum number of tasks in chains. This approach was undertaken to generate various instance shapes; from containing a small number

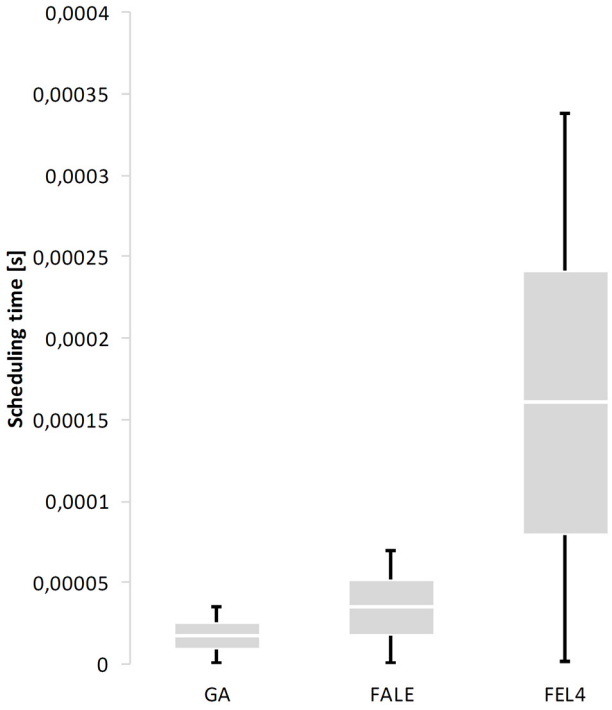

FIGURE 5. Average computation times for different numbers of tasks.

of long chains up to a large number of short chains. For each instance the maximum chain length was computed; this value varies from 1 (each chain contains only one task) up to the instance size (one chain contains all of the tasks).

Additionally, each instance containing less than 100 tasks was scheduled 10 000 times, while the larger instances were scheduled 1 000 times. The algorithm computation time was computed as the average of computing times for each instance. This approach increases the accuracy of scheduling times measurement.

The experiments were conducted on Intel IA-32 PC, equipped with 2.53GHz Core2Duo processor, but just a single core was used in our experiment. The operating system was Ubuntu Linux 10.04. All scheduling algorithms were implemented in $C++$.

5.2. Computation time results

The first considered parameter was the computation time needed to construct a schedule. Since all examined algorithms are polynomial time algorithms, the authors were able to construct solution for all generated instances and all algorithms. In Figure 5, the average scheduling times for each instance size, are presented.

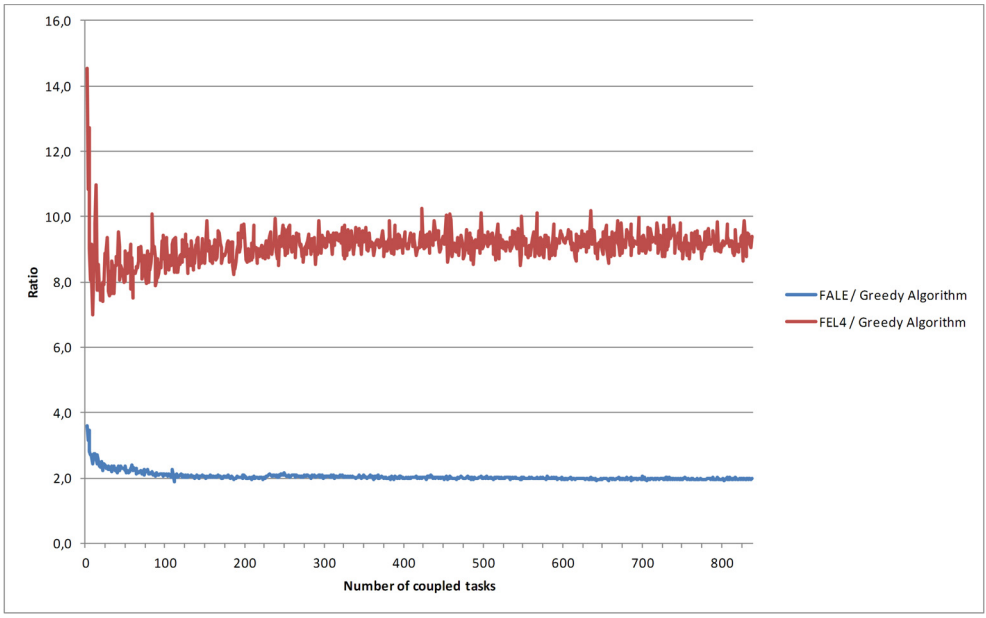

FIGURE 6. Ratio of the FALE and FEL4, the computation time of the greedy algorithm.

As expected, the greedy algorithm was proved to be the fastest for each instance. Moreover, the number of chains and their lengths had negligible impacts on its computation times. The FALE algorithm provided similar results - computation times are also proportional to the number of coupled tasks with also negligible impact of the number of chains and their lengths. On the other hand, in overall performance this algorithm is approximately two times slower than the greedy algorithm, except for very small instances.

The third analyzed algorithm – the FEL4 – has also execution time proportional to the number of tasks, but the number of chains and tasks in each chain has more significant impact on the computation time. This is clearly noticeable in Figure 6. Figure 6 shows the ratio of the computation time of FALE and FEL4 algorithms to that of the greedy algorithm, respectively.

It was interesting to observe that the computation time of the exact algorithm FEL4 tends to be 9 times longer than the fastest greedy algorithm if the size of the instance goes to infinity. This result indicates that FEL4 is fast enough to be implemented in practical systems in the situations where the optimal solution is required.

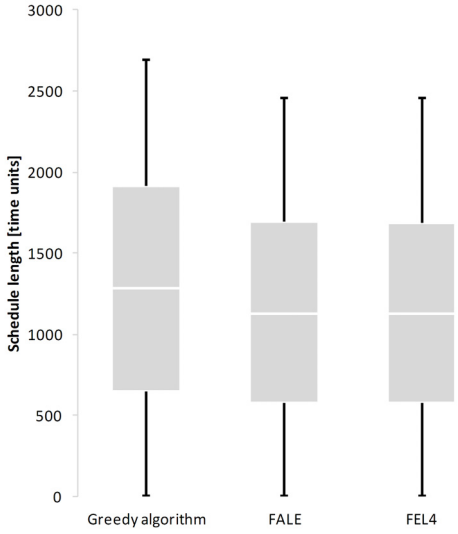

FIGURE 7. Average schedule lengths for different methods.

5.3. QUALITY OF THE SCHEDULES

The second analyzed parameter was the makespan (thus the quality) of solutions provided by the analyzed algorithms. It is obvious that the greedy schedule will not always be the optimal one; this case is illustrated in Figure 2.

The average results achieved are presented in Figure 7.

In Figure 7, the difference between the greedy algorithm and other algorithms can be clearly seen. The FEL4 was proved to provide optimal solutions (see [15]), and the length of the FALE's solution was proved to be not longer than L units of time in comparison with the optimal solution. We see that the difference in length between the achieved and optimal solutions of the FALE algorithm does not depend on the number of tasks and thus relatively goes to zero if the size of the instance goes to infinity.

During the tests, the FEL4 and the FALE algorithms constructed schedules similar in terms of the makespan. For $L = 4$ the largest overhead of solutions generated by the FALE algorithm was 6 units of time, thus, the theoretical limit of 8 units of time was never attained in the test set. This largest overhead was observed in 227 from 16 569 instances, which is 1,4% of the test cases. On the other hand, the optimal solutions were found by FALE 11 989 times, which represents 72,4% of instances. The mean overhead of FALE is 0,77 units of time, which represents 0,21% of the makespan. The greedy algorithm, as expected, provided solutions usually longer than optimal; the deviations were from 0% up to 33% with the mean of 13,2%. In terms of time units, the solutions differ from optimal up to 554, with the average of 147,5 time units. The optimal results were found only

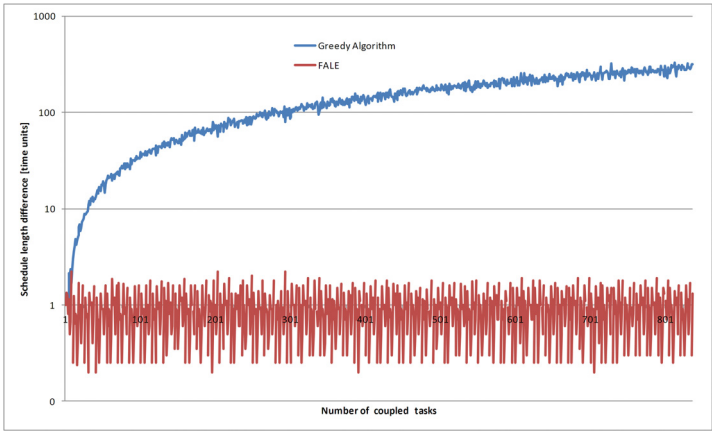

FIGURE 8. Average relative differences of schedule lengths obtained by FALE and greedy, respectively, as compared with the optimum.

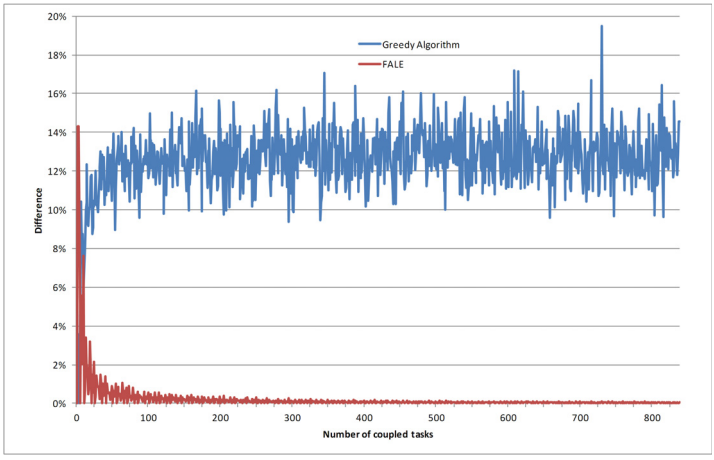

FIGURE 9. FALE and greedy percentage quality difference ratios.

in 3 505 out of 16 569 instances, that is in 21% of instances presented in Figure 8. Figure 9 shows the percentage quality ratio of these algorithms.

6. CONCLUSION

In the paper, the state of the art of the scheduling of the coupled tasks was presented. We have gathered the results of the research conducted in the last decade on that subject. The best known algorithms and the complexity results for

several types of the problems were presented. The recent literature was reviewed and main research development directions were emphasized.

What is more, the new experimental results for the lately constructed algorithms were presented. The experimental analysis presented in this paper shows that both algorithms FALE and FEL4 have a potential to be implemented in practical applications. The FALE approximation provides almost optimal solutions for most cases, with the cost of only two times greater computation time in comparison with the fastest but primitive greedy algorithm. What is more, the FALE algorithm provides significant improvement over the simple heuristics.

As far as the future research in this area is concerned, an interesting topic could be so called high multiplicity couple tasks scheduling, having applications in real-time systems.

Acknowledgements. The research was supported by Polonium Project 2011-2012 (No. 8406/2011) of the Polish Ministry of Science and Higher Education, French Ministry of Foreign and European Affairs and French Ministry of Higher Education and Research, and the grant of Polish National Science Center (No. 519 643340).

References

[1] A.A. Ageev and A.E. Baburin, Approximation algorithms for uet scheduling problems with exact delays. *Oper. Res. Lett.* **35** (2007) 533–540.
[2] D. Ahr, J. Bekesi, G. Galambos, M. Oswald and G. Reinelt, An exact algorithm for scheduling identical coupled tasks. *Math. Methods Oper. Res.* **59** (2004) 193–203.
[3] K. Baker, *Introduction to Sequencing and Scheduling.* J. Wiley, New York (1974).
[4] P. Baptiste, A note on scheduling identical coupled tasks in logarithmic time. *Disc. Appl. Math.* **158** (2010) 583–587.
[5] J. Bekesi, G. Galambos, M. Oswald and G. Reinelt, Improved analysis of an algorithm for the coupled task problem with uet jobs. *Oper. Res. Lett.* **37** (2009) 93–96.
[6] J. Blazewicz, P. Dell'Olmo, M. Drozdowski and M.G. Speranza, Scheduling multiprocessor tasks on three dedicated processors. *Inform. Process. Lett.* **41** (1992) 275–280.
[7] J. Blazewicz, M. Drabowski and J. Weglarz, Scheduling independent 2-processors tasks to minimize schedule length. *Inf. Process. Lett.* **18** (1984) 267–273.
[8] J. Blazewicz, K. Ecker, T. Kis, C.N. Potts, M. Tanas and J. Whitehead, Scheduling of coupled tasks with unit processing times. *J. sched.* **13** (2010) 453–461.
[9] J. Blazewicz, K. Ecker, T. Kis and M. Tanas, A note on the complexity of scheduling coupled tasks on a single processor. *J. Brazil. Comput. Soc.* **7** (2002) 23–27.
[10] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt and J. Weglarz, *Handbook of Scheduling. From Theory to Applications.* Springer Verlag (2007).
[11] J. Blazewicz, G. Pawlak and B. Walter, Scheduling production tasks in a two stage FMS. *Int. J. Prod. Res.* **40** (2002) 4341–4352.
[12] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Potts and J. Whitehead, Scheduling of coupled tasks and one-machine no-wait robotic cells. *Comput. Oper. Res.* **36** (2009) 301–307.
[13] P. Brucker, *Scheduling Algorithms.* Springer Verlag, Berlin, 3rd edition (2001).
[14] P. Brucker and S. Knust, Complexity results for single-machine problems with positive finish-start time-lags. *Osnabrück Schriften zur Mathematik* **202** (1998) 299–316.

[15] K. Ecker and M. Tanas, Complexity of scheduling of coupled tasks with chains precedence constraints and constant even length of the gap. *Found. Comput. Decision Sci.* **32** (2007) 199–212.
[16] K. Ecker and M. Tanas, Complexity of scheduling of coupled tasks with chains precedence constraints and constant even length of the gap. *J. Oper. Res. Soc.* **63** (2012) 524–529.
[17] M. Elshafei, H.D. Sherali and J.C. Smith, Radar pulse interleaving for multi-target tracking. *Naval Res. Logist.* **51** (2004) 79–94.
[18] A. Farina and P. Neri, Multitarget interleaved tracking for phased array radar. *IEEE Proc. Part F: Comm. Radar Signal Process* **127** (1980) 312–318,.
[19] R.L. Graham, E.L Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.* **5** (1979) 287–326.
[20] J.N.D. Gupta, *Single facility scheduling with two operations per job and time-lags.* Technical Paper (1994).
[21] J.N.D. Gupta, Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags. *J. Glob. Optim.* **9** (1996) 239–50.
[22] P.L. Heinselman, D.L. Preignitz, K.L. Manross anf T.M. Smith and R.W. Adams, Rapid sampling of severe storms by the national weather radar testbed phased array radar. *Weather Forecast.* **23** (2008) 808–824.
[23] A. Izquierdo-Fuente and J.R. Casar-Corredera. Optimal radar pulse scheduling using neural networks. In *IEEE International Conference on Neural Networks* **7** (1994) 4588–4591.
[24] V. Lehoux-Lebacque, N. Brauner and G. Finke, Identical coupled tasks scheduling: polynomial complexity of the cyclic case. *Les Cahiers Leibnitz* **179** (2009).
[25] J. Leung, editor. *Handbook of Scheduling.* Chapman and Hall (2004).
[26] R. McNaughton, Scheduling with deadlines and loss functions. *Manage. Sci.* **6** (1959) 1–12.
[27] A.J. Orman and C.N. Potts, On the complexity of coupled tasks scheduling. *Disc. Appl. Math.* **72** (1997) 141–154.
[28] A.J. Orman, C.N. Potts, A.K. Shahani and A.R. Moore, Scheduling for the control of a multifunctional radar system. *Eur. J. Oper. Res.* **90** (1996) 13–25.
[29] A.J. Orman, A.K. Shahani and A.R. Moore, Modelling for the control of a complex radar system. *Comput. Oper. Res.* **25** (1998) 239–249.
[30] C.N. Potts and J.D. Whitehead, Heuristics for a coupled-operation scheduling problem. *J. Oper. Res. Soc.* **58** (2007) 1375–1388.
[31] R.D. Shapiro, Scheduling coupled tasks. *Nav. Res. Logist. Quart.* **27** (1980) 477–481.
[32] M.I. Skolnik, *Introduction to Radar Systems.* McGraw Hill, London (1980).
[33] M. Tanas, *Scheduling of Coupled Tasks.* PhD thesis, Technische Universitat Clausthal (2004).
[34] J.D. Whitehead, *Scheduling and layout in flexible manufacturing systems.* Ph.D. thesis, University of Southampton (2002).
[35] W. Yu, *The Two-machine Flow Shop Problem with Delays and the One-machine Total Tardiness Problem Ph.D. Thesis.* Technische Universiteit Eindhoven (1996).
[36] 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. *J. Sched.* **7** (2004) 333–348.