Scheduling High Multiplicity Coupled Tasks

Wojciech Wojciechowicz, Michaël Gabay

FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 45 (2020) No. 1
ISSN 0867-6356
DOI: 10.2478/fcds-2020-0004 e-ISSN 2300-3405

sciendo

Scheduling High Multiplicity Coupled Tasks

Wojciech Wojciechowicz *, Michaël Gabay †

Abstract. The coupled tasks scheduling problem is class of scheduling problems, where each task consists of two operations and a separation gap between them. The high-multiplicity is a compact encoding, where identical tasks are grouped together, and the group is specified instead of each individual task. Consequently the encoding of a problem instance is decreased significantly. In this article we derive a lower bound for the problem variant as well as propose an asymptotically optimal algorithm. The theoretical results are complemented with computational experiment, where a new algorithm is compared with three other algorithms implemented.

Keywords: Coupled tasks, scheduling, complexity theory, asymptotically optimal algorithms, high multiplicity

1. Introduction

The coupled tasks scheduling problem was introduced by Shapiro in [10]. A task i is called coupled if it is a two operations job: there is a first operation of duration a_i and a second operation of duration b_i. Those two operations are separated by a fixed duration L_i (see Figure 1 for an example of coupled task). Obviously, operations cannot overlap, however, other operations can be processed during the idle time L_i. Shapiro in [10] proved that the coupled tasks scheduling problem is $\mathcal{NP}$-complete. In the problem considered in this work all tasks have to be scheduled on a single machine and the objective is to minimize the makespan.

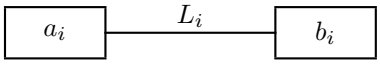

Figure 1: A single coupled task

*Institute of Computing Science, Poznań University of Technology, Poznań, wojciech.wojciechowicz@cs.put.poznan.pl
†University Grenoble Alpes, CNRS, G-SCOP, 38000 Grenoble, France

The coupled tasks scheduling problem was originally introduced for beam steering software for sophisticated radar devices [10]. In those systems, the first operation is to send a beam, and the second is to receive reflected pulse. In the mean time, the device is idle (which correspond to gap) and can be used to process other beam. The popularity of phased array radars for non-military purposes grows in the last years; such devices are more affordable on the market since modernisations in army provided still functional second-hand devices [3]. Notable example of such usage is monitoring of violent weather phenomena, e.g. hurricanes and tornadoes [7]. The High multiplicity scheduling problems could be found at repetitive manufacturing environments [4]. In this article we consider the case, where all tasks can be partitioned into small groups sharing the same properties. We assume, that the only difference between families is the gap size (where $\forall i \lfloor L_i \rfloor = L$, and $L_1 \leq L_2 \leq \cdots \leq L_i$, for ease of presentation). An example of such constraints can be found in mass production system, e.g. in paint shop where products can be grouped by color. The times needed to paint a car with chosen color is constant, but the waiting time between placing two layers depends on the color chosen.

1.1. Notation

The $\alpha|\beta|\gamma$ notation is commonly used to describe the coupled tasks scheduling problem, as introduced by Graham et al. [6] and further extended by Błażewicz et al. [2]. The α characterized processors in system. In this work we consider a single machine system, thus $\alpha = 1$.
The β describes the set of tasks and additional resources. In this work the case (p, L_i, p) is considered:

- p - the processing time of first and second operation,
- L_i - the length of the i^{th} coupled tasks gap,

And the γ field describes the optimality criterion. This work focused on the C_{max} criterion - schedule length or makespan, which is the completion time of last processed operation in schedule.

Bellow is a brief summary of the notation used in this work:

- n - the number of coupled tasks,
- i - the index used to represent the i^{th} coupled task,
- k - the number of coupled tasks families,
- q - the index used to represent the q^{th} family,
- m - the number of idle operations in schedule,
- block - a subsequence of operations starting with a_j^i and finishing with b_l^h, where a_l^h is the last operation a finishing before b_j^i starts (included).

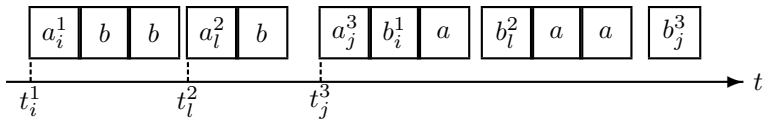

Figure 2: A complete block

- complete block - a block containing maximum number of operations (namely $2\lfloor L_i \rfloor + 2$ operations when $\forall_i \lfloor L_i \rfloor = L$, $a = b = 1$)
- window - a subsequence of operations starting with a_j^i and finishing with b_j^i

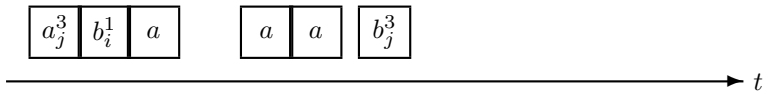

Figure 3: A window

- complete window - a window containing maximum number of operations (namely $\lfloor L_i \rfloor + 2$ operations)

1.2. Problem formulation

We consider the problem where all coupled tasks have identical processing times ($\forall i\ a_i = b_i = p$). Using Graham notations, we can denote the problem as $1|(p, L_i, p)\text{-}coupled, exact\ gap|C_{max}$, as proposed in [1].

Orman and Potts [9] proved that the problem $1|(p, L_i, p)\text{-}coupled, exact\ gap|C_{max}$ is $\mathcal{NP}$-hard in the strong sense whereas $1|(p, L, p)\text{-}coupled, exact\ gap|C_{max}$ is polynomial.

In our problem, there are k families $\{f_q\}_{q=1,\ldots,k}$ of coupled-tasks. Each family f_q contains n_q identical tasks with the same separation gap L_q. Remark that as a_q's and b_q's are all fixed and equal to p, the only characteristic that differs between the tasks families is L_q.

The problem $1|(p, L_i, p)\text{-}coupled, exact\ gap, families|C_{max}$ considered in this work can be stated as follows:

- there is a single processor in the system
- all tasks are coupled,
- all tasks within family are identical,
- all gaps are exact,
- processing time of each operation is equal to p,
- there are $k = 2$ families of tasks,
- all gaps within family are identical,

- all gaps have the same floor value,
- there is no precedence constraints,
- preemptions are not allowed,
- the optimisation criteria is C_{max} minimisation.

We can state the corresponding decision problem as follows:

Input: C, p and k pairs (L_q, n_q), $L_q \in \mathbb{Q}_+$ and $n_q \in \mathbb{N}_+$.
Output: **YES** and a certificate if there exists a solution to the problem $1|(p, L_i, p)$-*coupled*, *exact gap, families*$|C_{max}$ with makespan $\leq C$. **NO** otherwise.

Let $M = \max(\max_q(L_q), \max_q(n_q))$ and $n = \sum_{q=1}^{k} n_q$. Remark that the size of this problem is $O(k \log(M))$, which $-$ in most cases $-$ is $o(n)$. Hence a whole schedule cannot be given as a certificate because its size is an $O(n)$ which is exponential in $k \log(M)$. Such a problem is called a high multiplicity scheduling problem (see [8, 4, 5]) and even proving that such problems belong to $\mathcal{NP}$ is very challenging.
Note that $1|(p, L, p)$-$coupled, exact\ gap|C_{max}$ is a high multiplicity scheduling problem too. Yet, Orman and Potts [9] proved optimal placement and the optimal makespan of such an instance can easily be computed through a few simple operations.

$1|(p, L_i, p)$-*coupled*, *exact gap*$|C_{max}$ can be modeled as $1|(p, L_i, p)$-*coupled*, *exact gap, families*$|C_{max}$ using n families. Hence, we know that this problem is $\mathcal{NP}$-hard in the strong sense for the general case. Yet, when there is a small number of families, this is a whole different problem and we can derive results for such cases.

If we admit rational values of L_i's, we can divide all L_i's, a_i's and b_i's by p and our problem is now $1|(1, L'_i, 1)$-$coupled, exact\ gap, families|C_{max}$ with rational values of L_i's. For the ease and clarity of presentation in this article we will assume that $p = 1$, since both problems are equivalent.

In this article we consider a variation of scheduling high-multiplicity coupled tasks problem, where the gaps have the same floor values, thus:

$$\lfloor L_1 \rfloor = \lfloor L_2 \rfloor = \cdots = \lfloor L_k \rfloor = L$$

1.3. Lower bound

Orman and Potts proved in [9] that problem
$1|(a_i = b_i = p, L_i = L) - coupled, exact\ gap|C_{max}$ is polynomial. The algorithm they provided construct a schedule constructed from two parts - complete blocks followed by left-shifted scheduled remaining tasks. The length of the latter part is equivalent to the length of complete block minus number of tasks missing to construct a complete block (the time slots for those b operations will remain idle). Consequently, since:

- number of blocks - $\lceil \frac{n}{L+1} \rceil$
- length of a complete block - $2L + 2$

- number of tasks in complete block - $(L+1)$
- number of tasks, which remains to schedule after constructing complete blocks - $(n)mod(L+1)$

The makespan of such a schedule, as provided by Orman and Potts is:

$$C_{max} = \lceil \frac{n}{L+1} \rceil (2L+2) - ((L+1) - (n)mod(L+1)) \tag{1}$$

The length of a left-shifted block is determined by length of both operations, number of operations and a length of longest gap used to construct a block. A length of complete block is:

$$C = a + b + \lfloor L \rfloor + max(L_k) \tag{2}$$

where a and b are the length of first and second operation respectively. Since in the problem considered in this work $\forall_i \lfloor L_i \rfloor = L$, $a = b = 1$ and the pre-emptions are forbidden there are at most L operations to be scheduled during any gap. Otherwise an operation will collide with operation a or b of the task, which gap is filled. Consequently, the shortest completed block is the one constructed using tasks from family with smallest L_i and the length is:

$$C = 2 + \lfloor L \rfloor + max(L_k) \tag{3}$$

Given an instance, we only allow paying smaller gaps, hence all block lengths are l_1 and last block's length is $r + L_1 + 1$, where r is the number of remaining tasks to be scheduled. Since there is at least one task 2 to place, at some point an operation will be missing into a block. Let m be the total number of missing operations in all blocks. Since a block contains a minimum of 1 operation, the maximum value of m is L, which can holds up to $\lfloor L \rfloor$ operations. Otherwise the block is empty or collisions occur.

To place a L_2 within a block of type L_1 there is a need to shift tasks by $L_2 - L_1$, otherwise at least two operations will collide. Since pre-emptions are not allowed to achieve this there is a need to use additional $L_2 - L_1$ space on top of operation lengths, by re-arranging IDLE slots within a block. Because operation length is bigger than $L_2 - L_1$, it is sufficient to use one "'missing operation"' from incomplete block to place at most $\lfloor L-1 \rfloor$ L_2's. Consequently at most $\lfloor L-1 \rfloor$ m tasks from other than L_1 family can be placed within shorter blocks in any feasible schedule.

To conclude, the lower bound for the problem considered in this work is:

$$C_{max} \geq \begin{cases} \lfloor \frac{n}{L+1} \rfloor (2+L+L_1) + (nmod(L+1)+L) \ iff \ n_2 \leq m(L-1) \\ \lfloor \frac{n}{L+1} \rfloor (2+L+L_1) + (nmod(L+1)+L) + (L_2 - L_1) \lceil (\frac{n_2 - m(L-1)}{L+1}) \rceil \ otherwise \end{cases} \tag{4}$$

2. Block Schedule Algorithm

First, we introduce the Block Schedule Algorithm (BSA). The BSA is a very easy and intiutive asymtotically optimal algorithm. It is based on observation, that any

feasible schedule can be built from blocks. The minimum length of block is determined number of tasks used to construct a block, and length of the longest gap. The number of tasks with largest gap does not influence the block length - blocks with 1 or $L-1$ tasks with the largest gap have exactly the same length.
The BSA algorithm groups tasks in families (based on the length of gap) and construct complete blocks in each category (using $L+1$ tasks). Consequently at most L tasks of each family remains (otherwise it would be possible to construct a complete block, using $L+1$ tasks). Since we have two families, the remaining tasks can be arranged into at most two blocks, which we does in the last step, using remaining tasks. In case there are sufficient idle time slots, both block of type "a" is constructed, where type "b" tasks are fitted using idle slots. An example of such construction is given in the following figure.

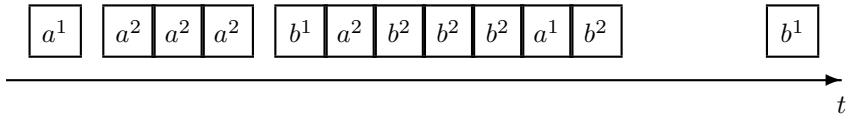

Figure 4: Last blocks

Using above properties we proposed the Block Schedule Algorithm. We define the *Block Schedule* as the schedule created by Algorithm 1.

The Block Schedule Algorithm only uses simple arithmetic, its complexity is $\mathcal{O}(nlogn)$.

2.1. Worst case analysis

As shown in 1.3 the lower bound for the problem is:

$$C_{max} = \begin{cases} \lfloor \frac{n}{L+1} \rfloor (2+L+L_1) + (n mod (L+1) + L) \ iff \ n_2 \leq m(L-1) \\ \lfloor \frac{n}{L+1} \rfloor (2+L+L_1) + (n mod (L+1) + L) + (L_2 - L_1) \lceil (n_2 - \frac{m(L-1)}{L+1}) \rceil \ otherwise \end{cases} \tag{5}$$

The GBSA constructs schedule from blocks, by grouping them into blocks and mixing the remainding part to opitmise the use of IDLE time slots. Since it is a heuristics it don't guarantee the optimiality of IDLE slots usage. There exists schedules, where GBSA uses longer blocks even if shortest were feasible. An example of such a case is given in 5. The upper schedule was constructed using GBSA algorithm while the lower one is the optimal schedule.

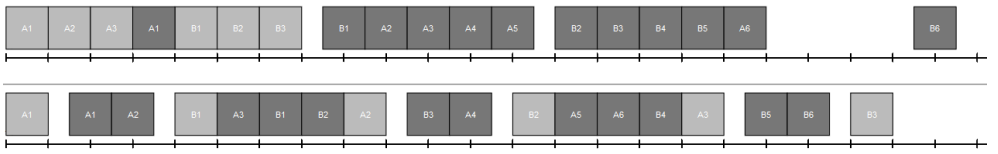

Figure 5: BSA - worst case example

Algorithm 1 Block Schedule

Input: k families of tasks (n_i, L_i) such that $\forall i \ \lfloor L_i \rfloor = L$ and $L_i \leq L_{i+1}$
Output: The makespan of the Block Schedule

1:
2: C $= 0$ {Initialize makespan}
3: **for** $i = 1 \to k$ **do**
4: $h \leftarrow \lfloor \frac{n_i}{L+1} \rfloor$
5: $n_i \leftarrow n_i - h(L+1)$
6: $C \leftarrow C + h(L + L_i + 2)$ {Make h complete blocks of tasks i}
7: **end for**
8:
9: $num \leftarrow 0$
10: **for** $i = 1 \to k$ **do**
11: **if** $n_i = 0$ **then**
12: **continue**
13: **end if**
14: $num = num + n_i$
15: **if** $num \geq L + 1$ **then**
16: {Creates a complete blocks, left shifted, with some of the remaining tasks}
17: $C \leftarrow C + (L + L_i + 2)$
18: $num \leftarrow num - (L + 1)$
19: **end if**
20: **end for**
21: **if** $num \neq 0$ **then**
22: {Creates an uncomplete blocks, left shifted, with the remaining tasks}
23: $C \leftarrow C + (num + L_i + 2)$
24: **end if**
25:
26: **return** C

As we have shown in 1.3, at most $\lfloor L-1 \rfloor m$ tasks from longer family can be placed within schedule constructed from shorter blocks only (by using m tasks from shorter family). In such a case, the GBSA algorithm will construct a $\lceil \frac{m(L-1)}{L+1} \rceil$ longer blocks and no shorter (since $m \leq L$), while the optimal schedule have all $\lceil \frac{m(L-1)}{L+1} \rceil$ blocks from shorter family. Since $m \leq L$ then maximum number of n_2 which can be hidden in shorter schedule is $L^2 - L$.

Let's denote C_{BSA} as a makespan of BSA and C_{opt} the optimal makespan and consider the worst-case (where $n_2 = L^2 - L$ and $n_1 = L$):

$C_{BSA} = \lceil \frac{(n_1+n_2}{L+1} \rceil (2+L+L_2) + ((n_1+n_2) mod (L+1) + L)$
while
$C_{opt} = \lceil \frac{n_1+n_2}{L+1} \rceil (2+L+L_1) + ((n_1+n_2) mod (L+1) + L)$
Consequently
$C_{BSA} - C_{opt} \leq (L_2 - L_1) \lceil \frac{n_1+n_2}{L+1} \rceil$
Since $n_2 = L^2 - L$ and $n_1 = L$ we have:
$C_{BSA} - C_{opt} \leq (L_2 - L_1) \lceil \frac{L+L^2-L}{L+1} \rceil$
and
$C_{BSA} - C_{opt} \leq (L_2 - L_1) \lceil \frac{L^2}{L+1} \rceil$

It is worth to mention, that the difference between optimal solution and the one produced by BSA does not depend on n but on the L which is not a part of instance size. Consequently, with the instance size going to infinity the BSA makespan goes to the optimality.

3. Computational experiment

In this section the results of computational experiment are provided. The analysis focus on two aspects the makespan and time needed to construct a schedule.

3.1. Test environment

The experiments were conducted on a platform with 8GB DDR3 RAM and Intel Core i5-3230M CPU (2.6 GHz), but only one core was used in computation. All algorithms were implemented in Java.

The test data was generated using custom task generator. Instances have two families with from 2 up to 99 tasks. To test various segment shapes, for each instance size (thus number of tasks from both families) all possible combination of n_1 and n_2 were generated multiple times. In total, 24480 test sets were examined.

3.2. 'Algorithm 2' by Orman and Potts

Orman and Potts in [9] proved, that the problem
$1|a_i = b_i = p, L_i = L|C_{max}$

is polynomial. They proposed an algorithm, which generates optimal schedule in $\mathcal{O}(nlogn)$ time. Since in the problem considered in this work $\lfloor L_i \rfloor = L$ we found reasonable to customize Orman and Potts Algorithm 2 and applied it to our problem. The algorithm divides schedule into two parts first constructed using complete blocks, then remaining tasks scheduled contiguously. The algorithm we applied creates complete blocks from each family. Remaining tasks are sorted ascending, and scheduled contiguously. In case more than one family is scheduled (which holds in the problem considered in this work), this algorithm does not guarantee schedule optimality, but still construct schedule in $\mathcal{O}(nlogn)$ time.

3.3. Greedy Algorithm

The Greedy Algorithm is a quick and intuitive algorithm. It assigns each task to first suitable position at schedule. Since there is no expensive calculation in this algorithm, it is very quick (runs in $\mathcal{O}n$ time) but obviously does not guarantee the solution optimality.

3.4. Full Search Algorithm

The Full Search Algorithm is an exact algorithm, which guarantees the optimality of the schedule by checking all potentially optimal (e.g. don't start with and idle, which obviously is unnecessary) feasible schedules. We construct the Full Search Algorithm based on a fact, that any feasible schedule can be represented as a sequence of tasks and idle times. Our implementation of Full Search Algorithm treats IDLE as a third family of tasks (just one operation) and permutes all tasks to construct a schedule. The usage of idle times provided means of reaching all feasible schedules. Consequently the search space is limited not only by the number of tasks, but also number of idle operations permuted.

A challenge we faced was to calculate the number of both number of idle time slots as well as idle time slot size. Since starting times might get any rational value we introduce intervals to classify values into equivalence classes. The goal is to establish points which have potential to change a schedule shape.

A schedule might be divided into slots. A slot size is a greatest common divisor of all entities lengths used in schedule. In our case, it is length of first operation, length of second operation and lengths of both first and second family gaps.

The number of idle slots (I) in a feasible schedule can be easily computed, e.g. based on makespan of Modified Orman and Potts' algorithm #2

$$I = C_{max} - 2 * n \tag{6}$$

To further reduce the search space we can divide the search space into slots as proposed above. Since Orman and Potts in [9] proved, that a makespan and its reverse problem are equivalent it is sufficient to consider only half of the slots in Full Search Algorithm's permutation.

4. Computational results

In this section the experimental results are presented. Two factors were analysed - the makespan, and the time needed to construct a schedule. We have followed the same approach as in [3]

4.1. Makespan

First, we focus on makespan. The Table 1 summarizes the average makespan per number of tasks.

Table 1: Average makespan

n	GBSA	OP	GA	FSA
2	7,50	7,50	7,50	7,50
3	8,50	8,50	8,65	8,50
4	9,50	9,50	9,67	9,50
5	10,50	10,50	13,53	10,50
6	16,60	16,90	16,97	16,10
7	17,50	17,83	18,02	17,33
8	18,64	18,79	19,16	18,50
9	19,69	19,75	21,27	19,69
11	26,85	27,15	27,55	26,15
12	27,77	28,09	28,65	27,41
13	28,88	29,04	29,95	28,71
14	29,92	30,00	32,65	29,92
15	30,96	30,96	35,01	30,96
16	37,10	37,40	38,05	36,20
17	37,91	38,34	39,55	37,59
18	39,12	39,29	41,25	38,92
19	40,17	40,25	43,87	40,17
20	41,21	41,21	45,90	41,21
25	51,46	51,46	56,97	NA
30	61,71	61,71	67,89	NA
35	71,96	71,96	78,86	NA
40	82,21	82,21	89,58	NA
45	92,45	92,45	100,86	NA
50	102,70	102,70	111,48	NA
55	112,95	112,95	122,72	NA
60	123,20	123,20	133,57	NA
65	133,45	133,45	144,41	NA
70	143,70	143,70	155,45	NA
75	153,95	153,95	166,67	NA
80	164,20	164,20	177,70	NA
85	174,45	174,45	188,22	NA
90	184,70	184,70	198,97	NA
95	194,95	194,95	210,21	NA
99	204,15	204,25	219,05	NA

The figure 6 illustrates the average makespans.

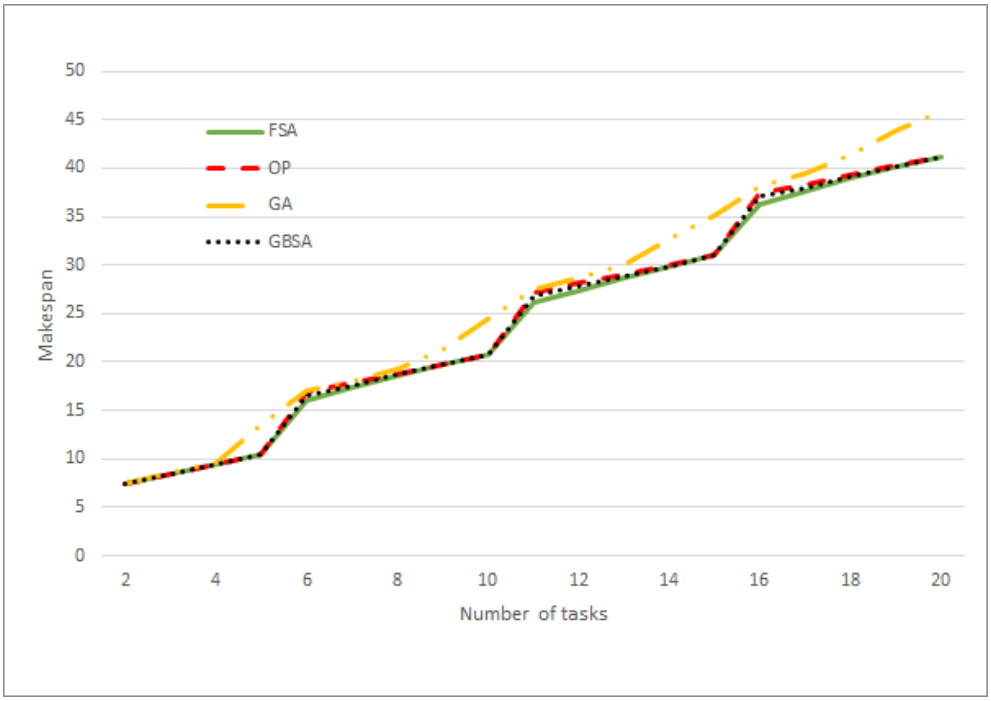

Figure 6: Schedules makespan

4.2. Scheduling times

The second aspect on basis of which the algorithms were evaluated was time needed to construct a schedule. The figure 7 illustrates the average scheduling time in milliseconds per number of tasks.

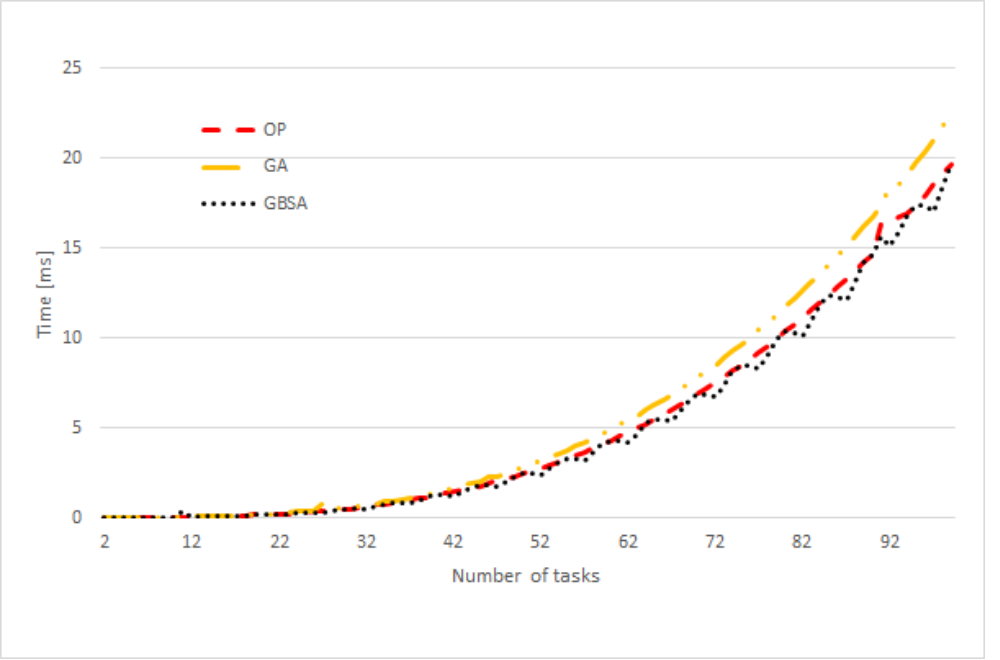

Figure 7: Scheduling times

As expected Greedy Algorithm, Orman and Potts' Algorithm as well as our Block Schedule Algorithm construct schedules in polynomial time. The BSA is slightly faster than the remaining algorithms, but the difference is negligible and could be mitigated by another implementation. The average computational times in seconds are provided in Table 2.

Table 2: Scheduling times [s]

n	GBSA	OP	GA	FSA
2	7.2×10^{-7}	2.8×10^{-7}	2.5×10^{-7}	5.5×10^{-5}
3	1.0×10^{-6}	6.2×10^{-7}	5.4×10^{-7}	3.6×10^{-5}
4	1.4×10^{-6}	9.4×10^{-7}	1.1×10^{-6}	2.8×10^{-5}
5	4.9×10^{-6}	1.6×10^{-6}	2.3×10^{-6}	1.0×10^{-5}
6	3.8×10^{-6}	3.6×10^{-6}	4.1×10^{-6}	8.6×10^{-3}
7	5.2×10^{-6}	8.0×10^{-6}	7.7×10^{-6}	1.1×10^{-2}
8	8.9×10^{-6}	1.1×10^{-5}	1.0×10^{-5}	9.7×10^{-3}
9	1.2×10^{-5}	1.2×10^{-5}	1.5×10^{-5}	7.0×10^{-3}
11	3.5×10^{-4}	4.2×10^{-5}	7.5×10^{-5}	7.7
12	4.7×10^{-5}	5.0×10^{-5}	7.0×10^{-5}	5.0
13	5.3×10^{-5}	4.4×10^{-5}	6.8×10^{-5}	3.6
14	5.5×10^{-5}	5.5×10^{-5}	6.7×10^{-5}	1.9
15	6.0×10^{-5}	7.4×10^{-5}	9.0×10^{-5}	5.2×10^{-1}
16	6.9×10^{-5}	8.0×10^{-5}	9.4×10^{-5}	2.4×10^{3}
17	7.5×10^{-5}	1.0×10^{-4}	1.2×10^{-4}	1.0×10^{3}
18	9.9×10^{-5}	1.1×10^{-4}	1.4×10^{-4}	4.9×10^{2}
19	1.5×10^{-4}	1.4×10^{-4}	1.5×10^{-4}	2.3×10^{2}
20	1.5×10^{-4}	1.5×10^{-4}	1.9×10^{-4}	4.7×10^{1}
25	2.8×10^{-4}	2.8×10^{-4}	3.3×10^{-4}	NA
30	5.0×10^{-4}	5.0×10^{-4}	5.8×10^{-4}	NA
35	8.2×10^{-4}	8.2×10^{-4}	9.3×10^{-4}	NA
40	1.2×10^{-3}	1.2×10^{-3}	1.4×10^{-3}	NA
45	1.8×10^{-3}	1.8×10^{-3}	2.0×10^{-3}	NA
50	2.5×10^{-3}	2.4×10^{-3}	2.8×10^{-3}	NA
55	3.3×10^{-3}	3.3×10^{-3}	3.8×10^{-3}	NA
60	4.3×10^{-3}	4.3×10^{-3}	4.9×10^{-3}	NA
65	5.5×10^{-3}	5.5×10^{-3}	6.2×10^{-3}	NA
70	6.9×10^{-3}	6.9×10^{-3}	7.8×10^{-3}	NA
75	8.5×10^{-3}	8.5×10^{-3}	9.6×10^{-3}	NA
80	1.0×10^{-2}	1.0×10^{-2}	1.2×10^{-2}	NA
85	1.2×10^{-2}	1.2×10^{-2}	1.4×10^{-2}	NA
90	1.5×10^{-2}	1.5×10^{-2}	1.7×10^{-2}	NA
95	1.7×10^{-2}	1.7×10^{-2}	2.0×10^{-2}	NA
99	2.0×10^{-2}	2.0×10^{-2}	2.2×10^{-2}	NA

The average times for Full Search Algorithm are not as trivial as one might expected. Although the algorithm is exponential, the growth rate is not proportional to the number of tasks. That is due to the fact, that the search space grows not only with the number of tasks, but also number of idle slots in schedule (detailed explanation is provided in 3.4). Since adding an idle is less effort-consuming than adding an task, the task / idle ratio has also an impact on the average scheduling time. The figure 8 illustrates the growth of FSA's scheduling time in search space.

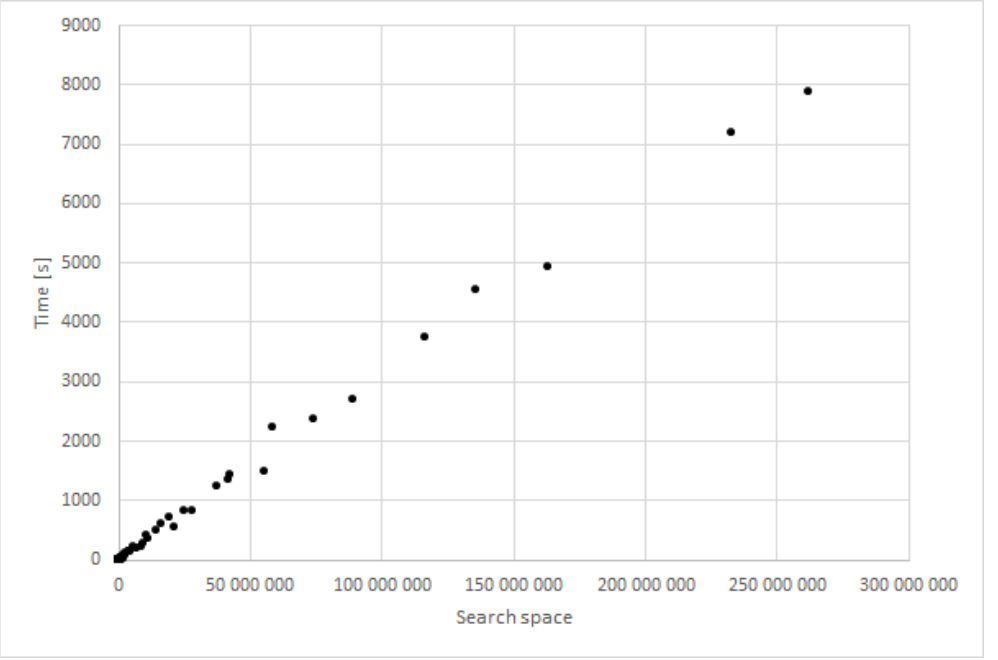

Figure 8: Full Search Algorithm - scheduling time vs search space

5. Conclusions

In this work the high multiplicity scheduling problem was considered. The authors focused on case, where all tasks can be partitioned into small groups sharing the same properties, thus $\forall i \lfloor L_i \rfloor = L$. The lower bound was provided and a new asymptotically optimal algorithm was proposed. The theoretical results was complemented with computational experiments, where performance (in terms of both computational times and the makespan) was compared with three other algorithms, namely Full Search Algorithm, Greedy Algorithm and customised Orman and Potts algorithm. The GBSA in most cases have found the optimal solution, while the computational time was polynomial in number of tasks.

Acknowledgments

The authors would like to dedicate their work on the Scheduling High Multiplicity Coupled Tasks to the memory of Gerd Finke. Professor Finke was a brilliant scientist, a restless researcher and the most patient and benevolent professor. Over his career, he contributed to numerous topics from theoretical mathematics to operations

research and scheduling. He animated the Operations Research community for years in Grenoble, in France and Europe and had friends and coauthors all over the world. The second author is grateful to Professor Finke for introducing him to scientific research (starting with the identical coupled tasks scheduling problem) and for giving him the chance to work with such a great man and learn by his side.

The research has been partially supported by the Polish-French bilateral programme POLONIUM (project no '8406/2011') and by statutory funds of Poznan University of Technology.

References

[1] Blazewicz J., Ecker K., Kis T., Potts C., Tanas M., and Whitehead J. Scheduling of coupled tasks with unit processing times. *Journal of Scheduling*, pages 1–9, 2010.

[2] Blazewicz J., Ecker K., Pesch E., Schmidt G., and Weglarz J. *Handbook of Scheduling. From Theory to Applications.* Springer Verlag, 2007.

[3] Blazewicz J., Pawlak G., Tanas M., and Wojciechowicz W. New algorithms for coupled tasks scheduling - A survey. *RAIRO - Operations Research - Recherche Opérationnelle*, 46:335–353, 2012.

[4] Brauner N., Crama Y., Grigoriev A., and van de Klundert J. A framework for the complexity of high-multiplicity scheduling problems. *Journal of combinatorial optimization*, 9(3):313–323, 2005.

[5] Clifford J. and Posner M. Parallel machine scheduling with high multiplicity. *Mathematical programming*, 89(3):359–383, 2001.

[6] Graham R., Lawler E., Lenstra J., and Kan A. R. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[7] Heinselman P., Preignitz D., anf T.M. Smith K. M., and Adams R. Rapid sampling of severe storms by the national weather radar testbed phased array radar. *Weather Forecasting*, 23:808–824, 2008.

[8] Hochbaum D. and Shamir R. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, pages 648–653, 1991.

[9] Orman A. and Potts C. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(1-2):141–154, 1997.

[10] Shapiro R. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27(3):489–498, 1980.

Received 29.10.2018, Accepted 24.01.2020