BATCH PROCESSING WITH SEQUENCE DEPENDENT SETUP TIMES: NEW RESULTS

PETR VILÍM

Charles University, Faculty of Mathematics and Physics, Malostranské náměstí 2/25, Praha 1, Czech Republic, e-mail: vilim@kti.mff.cuni.cz

Abstract: A constraint propagation (filtering) turned out to be an efficient method how to reduce search space for backtrack-based algorithms. This paper brings several improvements of filtering methods for batch processing with sequence dependent setup times.

The first one is domain filtering based on a precedence graph—such filtering can be useful even when no precedences are given at the start. I describe how to easily detect all the precedences found by the edge-finding algorithm and how to maintain a transitive closure of the precedence graph in reasonable time. The results for the precedence graph hold also for the usual disjunctive scheduling (unary resource) because it is a special case of batch processing (without any setups).

This paper also proposes a new, more effective version of the not-first/not-last filtering algorithm for batch processing with sequence dependent setup times.

1 INTRODUCTION

The batch processing with sequence dependent setup times is an extension of classical disjunctive scheduling (for the definition of disjunctive scheduling, see for example [1]). The task is to schedule a set of activities T on one resource with the following characteristics:

- Each activity $i \in T$ has its *release time* r_i and *deadline* d_i. Processing of the activity cannot start before the release time r_i and cannot complete after the deadline d_i.
- Each activity $i \in T$ has a *family* (type) f_i. The set of all families is F.
- Only activities of the same family can be processed together. Activities processed together form a *batch*—their processing starts together and completes together.
- *Processing time* of an activity $i \in T$ depends only on the family of this activity f_i. Let p_{f_i} denote such processing time (or p_i for short). Processing of an activity cannot be interrupted (non-preemptive scheduling), once it starts it takes exactly p_i time units until it completes.
- The sum of all *capacities* c_i of the activities in one batch cannot exceed the capacity C of the resource (i.e. the resource is renewable).
- After a batch completes a *setup* is needed before another batch can start. The time of this setup depends on the families of both consequent batches. If the first batch is of family f and the second one is of family g, then s_{fg} denotes minimum setup time between them.

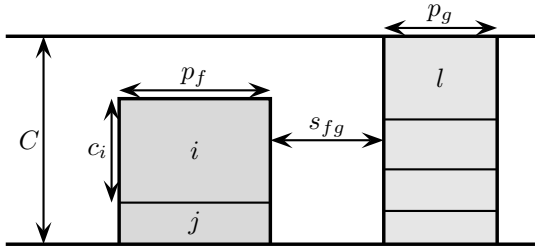


Figure 1: Example of two consecutive batches with families f and g.

Setup times have to satisfy the following two conditions. No setup is needed between two batches of the same family:

$$\forall f \in F : \; s_{ff} = 0$$

And the following triangle inequality holds for setup times:

$$\forall f, g, h \in F: \; s_{fg} + s_{gh} \geq s_{fh}$$

1.1 Formulation of the Problem in Constraint Programming

The described problem can be seen as a set of variables (i.e. starting time of each activity) and a set of constraints (i.e. all requirements for processing mentioned above). The task is to find an assignment to the variables which satisfies all constraints. This is the problem description for constraint programming (CP).

The usual way how to solve similar scheduling problems using CP is *constraint propagation*: take each constraint and try to recognize which values of involved variable are not possible (so-called *inconsistent values*). When there is only one possible value left for a given variable then we found the right assignment for this variable. The process of removing inconsistent values for a given constraint is often called *filtering*.

A good way to improve the constraint propagation is to use one (so-called *global*) constraint instead of several simple constraints. A Filtering algorithm for such global constraint can recognize more inconsistent values than the original constraints. In this paper, I consider only one global constraint for the whole resource and present several improvements of the filtering algorithms for this constraint. All of these algorithms remove inconsistent values by increasing the release times r_i or decreasing the deadlines d_i of the activities.

Constraint propagation is often not sufficient to find the solution. In this case, one of the ways to find a solution is to use a search method (e.g. backtracking) and constraint propagation can be used as an efficient method form pruning the search space. This is the case of this paper.

There are four existing filtering algorithms for the problem of batch processing with sequence dependent setup times: *edge finding, not-first/not-last, not-before/not-after* (see [7]) and *sequence composition* (see [6]). Each of these algorithms filters out different inconsistent values, therefore they can be used together to get maximum pruning.

1.2 Basic Notations

All filtering algorithms in this paper deal with some subsets of activities $\Omega \subseteq T$, new notation is needed for such a set Ω. I use the same notation as in [7].

Consider any subset of the activities $\Omega \subseteq T$. Processing of the set Ω can start at first at the time

$$r_\Omega = \min\{r_i, i \in \Omega\}$$

and cannot end after the time

$$d_\Omega = \max\{d_i, i \in \Omega\}$$

F_Ω denotes the set of all families in the set Ω:

$$F_\Omega = \{f, \, i \in \Omega \;\&\; f_i = f\}$$

Let $c(\Omega, f)$ be the sum of the capacities of all activities in the set Ω of family f, and let $u(\Omega, f)$ be the minimum time needed for processing these activities. Value of $u(\Omega, f)$ is computed simply from the minimum number of batches; setup times, release and due times are not taken into account.

$$c(\Omega, f) = \sum_{\substack{i \in \Omega \\ f_i = f}} c_i$$

$$u(\Omega, f) = \left\lceil \frac{c(\Omega, f)}{C} \right\rceil p_f$$

Minimum pure (i.e. without setups) processing time of the set Ω is then:

$$u(\Omega) = \sum_{f \in F_\Omega} u(\Omega, f)$$

Computation of minimum setup time needed for the processing of the set Ω is little bit more complicated. Again, release times and deadlines are not taken into account when computing this minimum setup $s(F_\Omega)$, this minimum setup time is only estimation based only on the families F_Ω.

Some algorithms also need the minimum setup time of processing of the set Ω under the condition that the first batch is of family f. Let $s(f, F_\Omega)$ be such minimum setup time. Similarly the $s(\Omega, f)$ is the minimum setup time needed for processing of the set Ω if the last batch is of family f. Let $k = |F|$ denote the number of families, function s can be precomputed in the time $O(k^2 2^k)$ for all subsets of families. Such computation is needed only once when the constraint is added into the system. If the number of activities k is a small number then this time complexity is tolerable. For the details how these functions can be computed see [7].

Finally the minimum processing time of the set Ω (including setup times) is:

$$p_\Omega = u(\Omega) + s(F_\Omega)$$

If the processing of the set Ω starts by an activity $i \in \Omega$ then the minimum processing time is:

$$p(i, \Omega) = u(\Omega) + s(f_i, F\Omega)$$

2 EDGE-FINDING

Filtering based on precedence graph has a tight relation to the edge-finding filtering algorithm, therefore I briefly resume it here. The whole algorithm can be found in [7].

Consider an arbitrary set $\Omega \subsetneq T$ and an activity $i \in (T \setminus \Omega)$. Assume for a while that the activity i is scheduled before the set Ω. Then the processing of the set Ω can start at first in the time $r_i + p_i$. The minimum setup time needed during the processing of the activities $\Omega \cup \{i\}$ is $s(f_i, F_\Omega \cup \{f_i\})$ because the activity i is scheduled first. If such processing ends after the deadline d_Ω then the assumption that the activity i can be scheduled before the set Ω, was wrong. This is the idea of the following rule *not-before*:

$$\forall \Omega \subset T,\ \forall i \in (T \setminus \Omega): \quad (1)$$
$$r_i + p_i + s(f_i, F_\Omega \cup \{f_i\}) + u(\Omega) > d_\Omega \Rightarrow i \not\ll \Omega$$

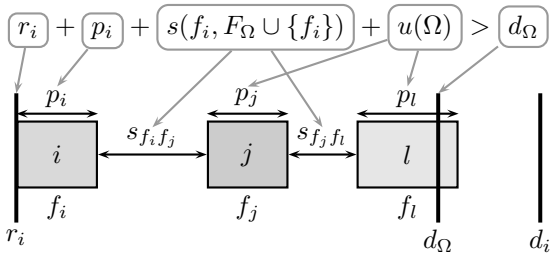


Figure 2: An example of the rule not-before for the activity i and the set $\Omega = \{j, l\}$.

Similarly, next rule says that when the activity i cannot be scheduled between the activities Ω it must be scheduled before the whole set Ω or after it:

$$\forall \Omega \subset T,\ \forall i \in (T \setminus \Omega): \quad (2)$$
$$d_\Omega - r_\Omega < p(\Omega \cup \{i\}) \ \Rightarrow\ (i \ll \Omega \text{ or } \Omega \ll i)$$

When both rules (1), (2) hold then the activity i has to be scheduled after the set Ω, i.e. $\Omega \ll i$. The resulting time bond adjustment is:

$$\Omega \ll i \ \Rightarrow \quad (3)$$
$$r_i \geq \max\{r_{\Omega'} + u(\Omega') + s(F_{\Omega'} \cup \{f_i\},\ f_i),\ \Omega' \subseteq \Omega\}$$

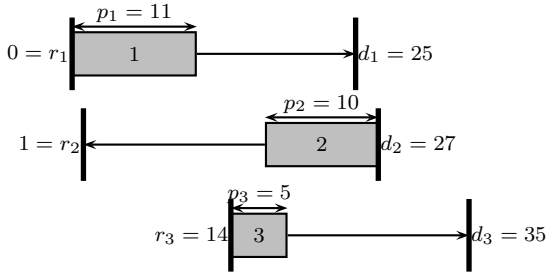
There are also symmetric rules for precedences $i \ll \Omega$. The edge-finding algorithm enforces all the reductions resulting from these rules. However, a repeated run of the algorithm can find further reductions because the time bounds have changed. Therefore it is suggested to repeat the edge-finding algorithm until no more reductions are found. Time complexity of one run of the algorithm [7] is $O(kn^2)$ where $n = |T|$ is the number of activities and $k = |F|$ is the number of families.

3 PRECEDENCE GRAPH

3.1 Motivation

In the following picture there is an example of a disjunctive scheduling problem. Disjunctive scheduling is a special case of batch processing: all setup times are 0, all capacities are 1 and the capacity of the resource is also 1. So in the case of disjunctive scheduling it is not possible to process more than one activity simultaneously.

None of the mentioned filtering algorithms (i.e. the algorithms edge-finding, not-first/not-last, not--before/not-after and sequence composition) is able to find any time bound adjustment for this problem:



Edge-finding recognizes that the activity number 1 must be processed before the activity number 3, i.e. $1 \ll 3$, and similarly $2 \ll 3$. Still, each of these precedences alone is weak—it does not enforce any change of the time bounds. However, from the knowledge $\{1, 2\} \ll 3$ we can deduce $r_3 = r_1 + p_1 + p_2 = 21$. Therefore a precedence-graph-based filtering can be useful even if no additional precedences are given at the start of the problem.

3.2 Detecting Precedences

The easiest way to find new precedences is the disjunctive constraint. It states that if an activity i cannot be processed before nor together with an activity j then it has to be processed after the activity j. In case of batch processing with sequence dependent setup times the activities i and j can be processed together iff:

$$f_i = f_j \quad \& \quad c_i + c_j \leq C$$
$$\& \quad \min\{d_i, d_j\} - \max\{r_i, r_j\} \geq p_i \quad (4)$$

And the activity i can be processed before the activity j iff:

$$r_j + p_j + s_{f_j f_i} \leq d_i - p_i \quad (5)$$

When the both conditions (4) and (5) do not hold then the precedence $i \ll j$ is called *detectable*.

Now I will show that all precedences found by the edge-finding algorithm can be subsequently easily found by the disjunctive constraint:

Proposition 1 *When edge-finding is unable to find further time bound adjustments then all precedences which the edge-finding found are detectable.*

Proof: Let us suppose that the edge-finding algorithm proved $\Omega \ll j$. I show that for an arbitrary activity $i \in \Omega$ the edge-finding algorithm made r_j big enough in order the precedence $i \ll j$ to be *detectable*.

Edge-finding proved $\Omega \ll j$ so the condition (2) holds:

$$\begin{aligned} p(\Omega \cup \{j\}) &> d_\Omega - r_\Omega \\ u(\Omega \cup \{j\}) + s(F_\Omega \cup \{f_j\}) &> d_\Omega - r_\Omega \\ d_\Omega - u(\Omega \cup \{j\}) - s(F_\Omega \cup \{f_j\}) &< r_\Omega \end{aligned} \quad (6)$$

Because edge-finding is unable to further change the time bounds according to (3):

$$r_j \geq \max\{r_{\Omega'} + u(\Omega') + s(F_{\Omega'} \cup \{f_j\},\, f_j),\ \Omega' \subseteq \Omega\}$$
$$r_j \geq r_\Omega + u(\Omega) + s(F_\Omega \cup \{f_j\}, f_j)$$

In this inequality, r_Ω can be replaced by the left side of the inequality (6):

$$\begin{aligned} r_j > d_\Omega &- u(\Omega \cup \{j\}) - s(F_\Omega \cup \{f_j\}) \\ &+ u(\Omega) + s(F_\Omega \cup \{f_j\}, f_j) \end{aligned}$$

And because:

$$\begin{aligned} u(\Omega) - u(\Omega \cup \{j\}) &\geq -p_i \\ s(F_\Omega \cup \{f_j\}, f_j) - s(F_\Omega \cup \{f_j\}) &\geq 0 \\ d_\Omega &\geq d_i \end{aligned}$$

the following inequality holds:

$$r_j > d_i - p_j$$

And thus neither condition (4) nor (5) holds and the precedence $i \ll j$ is *detectable*.

The proof for the precedences resulting from $j \ll \Omega$ is symmetrical. □

Filtering based on a precedence graph is simple: build up the Ω of all the activities which have to be processed before a given activity i and then use the rule (3):

Sort the activities according to r_i
for $i \in T$ **do begin**
  $\Omega := \emptyset$;
  $m := -\infty$;
  for $j \in T$ in non-increasing order of r_j **do**
    if $j \ll i$ **then begin**
      $\Omega := \Omega \cup \{j\}$;
      $m := \max\left(m,\ r_j + u(\Omega) + s(F_\Omega \cup \{f_i\},\, f_i)\right)$;
    end;
  $r_i := \max(m, r_i)$;
end;

The algorithm has time complexity $O(n^2)$. There is also a symmetric version of this algorithm for the precedences $i \ll \Omega$.

3.3 Transitive Closure

Several authors suggest to compute a transitive closure of the precedence graph (e.g. [2], [3]) to improve the filtering based on a precedence graph. But computing a transitive closure is time-consuming (for instance, time complexity of the Floyd-Warshall algorithm is $O(n^3)$). After the addition of a new precedence into the graph, the correction of the transitive closure can be made in the time $O(n^2)$. Still, the number of newly discovered precedences can be theoretically even $O(n^2)$.

In this section, I show that the correction of the transitive closure has to be made only after the adding a *non-detectable* precedence.

If the precedence $i \ll j$ is *propagated* by the edge-finding algorithm or by the precedence-graph-based filtering then the new time bounds fulfill the following inequalities (special cases of the rule (3) and its symmetric version for $\Omega = \{i\}$ and $\Omega = \{j\}$):

$$\begin{aligned} r_j &\geq r_i + p_i + s_{f_i f_j} \\ d_i &\leq d_j - p_j - s_{f_i f_j} \end{aligned}$$

Proposition 2 *Let $a \ll b$, $b \ll c$ and one of these precedences be detectable and the second one propagated. Then the precedence $a \ll c$ is detectable.*

Proof: We distinguish two cases:

1. $a \ll b$ is detectable and $b \ll c$ is propagated. Because the precedence $b \ll c$ is *propagated*:

$$r_c - s_{f_b f_c} \geq r_b + p_b$$

and because the precedence $a \ll b$ is *detectable*:

$$\begin{aligned} r_b + p_b + s_{f_b f_a} &> d_a - p_a \\ r_c - s_{f_b f_c} + s_{f_b f_a} &> d_a - p_a \end{aligned}$$

Together with the triangle inequality for the setup times $s_{f_b f_c} + s_{f_c f_a} \geq s_{f_b f_a}$:

$$r_c + s_{f_c f_a} > d_a - p_a$$

Thus neither of the conditions (4), (5) holds and the precedence $a \ll c$ is *detectable*.

2. $a \ll b$ is propagated and $b \ll c$ is detectable. Because the precedence $a \ll b$ is *propagated*:

$$d_b - p_b \geq d_a + s_{f_a f_b}$$

And because the second precedence $b \ll c$ is *detectable*:

$$\begin{aligned} r_c + p_c + s_{f_c f_b} &> d_b - p_b \\ r_c + p_c + s_{f_c f_b} &> d_a + s_{f_a f_b} \end{aligned}$$

We use the triangle inequality again, this time $s_{f_c f_a} + s_{f_a f_b} \geq s_{f_c f_b}$:

$$r_c + p_c > d_a - s_{f_c f_a}$$

Once again, neither of the condition (4), (5) holds and the precedence $i \ll j$ is *detectable*. □

According to the previous proposition: Let us suppose that the edge-finding algorithm or filtering based on a precedence graph do not yield any further reductions. Then a lot of precedences from the transitive closure are *detectable*, the only missing precedences (i.e. the precedences from transitive closure which are not *detectable* now) are in the transitive closure of the *non-detectable* precedences. Usually, only one *non-detectable* precedence is added in each search step (e.g. as a search decision) and so we can repair the transitive closure of *non-detectable* precedences in the time $O(n^2)$ (at each search step).

4 NOT-FIRST/NOT-LAST

The previous algorithm not-first/not-last [7] is based on the algorithm [1] and has time complexity $O(kn^2)$. In some situations, it uses only a weaker version of the filtering rules. Here I present a new version of this algorithm based on the algorithm from the article [4]. It doesn't use weaker rules and has better time complexity $O(n^2)$. However, more iterations of this algorithm may be needed before no more adjustments are found.

The not-first rule is based on the following idea. Consider an arbitrary set $\Omega \subsetneq T$ and an activity i which is not in the set Ω. Let us suppose that the activity i is processed before the set Ω or in the first batch of the set Ω. Then the processing of the activities $\Omega \cup \{i\}$ can start at first at the time r_i and has to be completed at latest at the time d_Ω. If there is not enough time for such processing then the assumption is wrong and the activity i can be processed at first after one batch of the set Ω:

$$r_i + p(i, \Omega \cup \{i\}) > d_\Omega \;\Rightarrow\; i \not\prec \Omega \qquad (7)$$

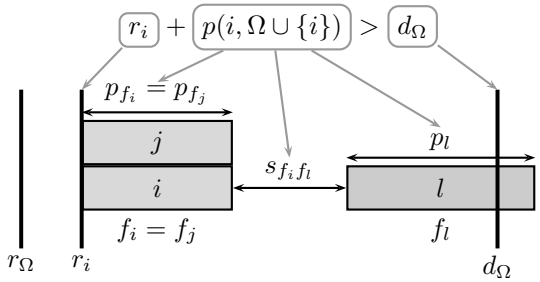


Figure 3: An example of the rule not-first for the set $\Omega = \{j, l\}$ and the activity i, $f_j = f_i$.

The fact that $i \not\prec \Omega$ allows us to change the release time of the activity i:

$$i \not\prec \Omega \;\Rightarrow\; r_i \geq \min\{r_j + p_j + s_{f_j f_i},\; j \in \Omega\} \quad (8)$$

It is clear that if there is some activity $j \in \Omega$ such that $r_j + p_j + s_{f_j f_i} \leq r_i$ then the set Ω cannot increase the release time r_i of the activity i. The only activities which can possibly increase the release time r_i of the activity i are:

$$\Psi = \{j,\; j \in T \;\&\; j \neq i \;\&\; r_j + p_j + s_{f_j f_i} > r_i\}$$

We are looking for some set $\Omega \subseteq \Psi$ for which the inequality (7) holds. The algorithm constructs the set Ψ by adding one activity after another. After each addition two values are recomputed:

$$\text{eftMin} = \min\{r_j + p_j + s_{f_j f_i},\; j \in \Psi\}$$
$$\text{lst} = \min\{d_\Omega - p(i, \Omega \cup \{i\}),\; \Omega \subseteq \Psi\}$$

If $\text{lst} < r_i$ then there exists a set $\Omega \subseteq \Psi$ for which the rule (7) holds but we do not know this set. However, we can use the rule (8) for the set Ψ—the drawback is that the resulting increase of the release time r_i can be lower. But if there exists any set for which (7) holds and (8) increases the value of the release time r_i then the algorithm increases this release time. Therefore if the algorithm is repeated until no more reductions are found then all the time bound adjustments resulting from the rules (7) and (8) are made.

Sort the activities according to d_i
for $i \in T$ **do begin**
  eftMin := ∞ ;
  lst := ∞ ;
  Ψ := $\emptyset$;
  for $j \in T$ in nondecreasing order of d_j **do**
    if $i \neq j$ **and** $r_j + p_j + s_{f_j f_i} > r_i$ **then begin**
      Ψ := $\Psi \cup \{j\}$;
      eftMin := $\min\left(\text{eftMin}, r_j + p_j + s_{f_j f_i}\right)$;
      lst := $\min\left(\text{lst}, d_\Psi - p(i, \Psi)\right)$;
      if lst $< r_i$ **then begin**
        r_i := eftMin;
        break;
      end;
    end;
end;

The time complexity of this algorithm is $O(n^2)$.

5 EXPERIMENTAL RESULTS

All filtering algorithms were combined into one algorithm in the following way:

repeat
  repeat
    repeat
      repeat
        consistency check
        not-first/not-last
      until no more changes found
      precedence graph
    until no more changes found
    edge-finding
  until no more changes found
  not-before/not-after
until no more changes found

The order in which the algorithms are called is not important for the resulting filtering. The selected order is the fastest for the tested problems.

I used the benchmark set [5]. The search strategy which I used is simple: take the activity which can be scheduled first and schedule it in its release time. It is the same search strategy that I used in [7]. The table 1 shows the results (column new) and comparison with the previous results [7] (column old).

				new		old	
problem	n	k	solutions	backtracks	time	backtracks	time
a	30	3	88	12	1.24s	12	2.22s
b	25	5	56251	5016	16m 10s	5016	25m 30s
c	25	5	72	0	0.80s	0	1.78s
d	40	6	12	0	0.41s	1	1.02s
e	20	2	28	0	0.13s	0	0.20s
f	50	6	6	2	0.34s	2	0.70s
g	30	3	1690	77	21.96s	77	37.08s
h	75	5	12	2	1.38s	2	2.90s
i	50	5	48	44	3.44s	44	7.59s
j	50	5	10	4	0.69s	4	1.34s
k	50	7	9	0	0.72s	0	1.33s
l	50	5	4	0	0.20s	0	0.51s
m	50	3	3	3	0.16s	3	0.35s
n	30	5	39	13	0.79s	13	1.78s
o	50	5	32	8	1.62s	8	3.60s
p	30	3	270	18	3.53s	24	6.05s
q	50	7	228	0	12.44s	0	28.03s
r	50	7	324	0	22.79s	0	44.94s
s	100	7	50	0	11.31s	0	26.48s
t	200	7	8	0	7.50s	0	18.00s
v	100	7	240	0	54.26s	0	2m 6s
w	50	2	24	4	0.83s	4	1.36s
x	50	2	1368	384	39.57s	384	1m 8s
z	50	4	24	6	1.06s	7	2.14s

Table 1: Experimental results

6 CONCLUSIONS

The experimental results show that the improvements proposed in this paper reduce the running time by one half for the lot of problems. However, this speed-up is not caused by better filtering (only for the problems d, p and z the number of backtracks decreased). There are two reasons for the speed-up:

- The new version of the not-first/not-last algorithm is faster. (However, the number of its repetitions may have increased).
- The filtering based on a precedence graph is also fast and can make a lot of filtering which was done by edge-finding. Hence the slower edge-finding algorithm is not repeated so often.

References

[1] Philippe Baptiste and Claude Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group*, 1996.

[2] Peter Brucker. Complex scheduling problems, 1999. URL `http://citeseer.nj.nec.com/brucker99complex.html`.

[3] W. Nuijten F. Foccaci, P. Laborie. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the 4th International Conference on AI Planning and Scheduling, AIPS'00*, pages 92–101, 2000.

[4] Philippe Torres and Pierre Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 1999.

[5] P. Vilím and R. Barták. A benchmark set for batch processing with sequence dependent setup times, 2002. URL `http://kti.mff.cuni.cz/~vilim/batch`.

[6] P. Vilím and R. Barták. A filtering algorithm sequence composition for batch processing with sequence dependent setup times. Technical Report 2002/1, Charles University, Faculty of Mathematics and Physics, 2002.

[7] P. Vilím and R. Barták. Filtering algorithms for batch processing with sequence dependent setup times. In *Proceedings of the 6th International Conference on AI Planning and Scheduling, AIPS'02*, 2002.