

Filtering Algorithms for Batch Processing with Sequence Dependent Setup Times

Petr Vilím and Roman Barták

Charles University

Faculty of Mathematics and Physics

Malostranské náměstí 2/25, Praha 1, Czech Republic

`vilim@kti.mff.cuni.cz`,

`bartak@kti.mff.cuni.cz`

Abstract

Domain filtering is a powerful technique for reduction of the search space during solving combinatorial problems like scheduling. In this paper we present several filtering algorithms designed specifically for scheduling in batch processing environments with sequence dependent setup times. We extend two known algorithms, namely edge finding and not-first/not-last technique, and we present a new filtering algorithm called not-before/not-after. Each of these algorithms removes some inconsistencies that are not detected by the other two algorithms. Thus, all the algorithms are assumed to run together to achieve better pruning.

Introduction

Constraint programming (CP) is becoming a popular tool for solving large combinatorial problems. It provides natural modeling capabilities to describe many real-life problems via domain variables and constraints among these variables. There exist generic techniques for constraint satisfaction usually based on integration of search (enumeration, labeling) with constraint propagation (domain filtering). Moreover, the CP framework allows integration of problem-dependent filtering algorithms into the constraint solver. These filtering algorithms typically encapsulate a set of elementary constraints; then we are speaking about global constraints. By using semantic information the global constraint typically removes more inconsistencies than generic constraint propagation through the elementary constraints. Still the global constraints keep reasonable (polynomial) time and space complexity.

Many global constraints/filtering algorithms have been designed for various classes of scheduling problems. One of the most important filtering techniques for solving the disjunctive scheduling problems is the edge-finding algorithm (for definitions of disjunctive and cumulative scheduling see (Baptiste & Le Pape 1996)). This algorithm was originally proposed in (Carlier & Pinson 1989) and there are also other versions of this algorithm like (Martin & Shmoys 1996). Edge-finding can also be adapted for cumulative scheduling (Baptiste & Le Pape 1996). Not-first/not-last is another filtering algorithm for disjunctive scheduling described in (Baptiste & Le Pape 1996). This technique can be combined with edge-finding to achieve even better pruning.

Caseau and Laburthe (1994) proposed the scheduling technique called tasks intervals for both disjunctive and cumulative scheduling. However, the combination of edge-finding with not-first/not-last outperforms the tasks intervals. There are also special techniques designed for cumulative scheduling like elastic relaxations (Baptiste, Le Pape, & Nuijten 1998) or energetic reasoning.

In this paper, we concentrate on filtering algorithms for batch processing with sequence dependent setup times. Such environment is quite often in real-life scheduling problems where batches describe a collection of activities processed together, e.g. in the pool, and the set-up times describe the time necessary to prepare the resource for the next batch, e.g. to clean up the pool. Batch processing and setup times impose additional constraints to the problem and these constraints can be used to prune the search space. We propose to encapsulate these additional constraints into a global filtering algorithm that achieves better pruning than existing global constraints for scheduling. In particular this algorithm exploits information about overlapping of activities – in batch processing, either the activities do not overlap or if two activities are overlapping then they start and complete at the same times. Thus, in some sense batch processing can be seen as a mixture of cumulative scheduling with disjunctive scheduling. The filtering algorithms for cumulative scheduling are weak when applied to batch processing because they do not use information about batches. The stronger filtering algorithms for disjunctive scheduling cannot be applied directly to batch scheduling because of parallel activities in the batch. So we extended the existing algorithms for disjunctive scheduling, namely edge finding and not-first/not-last, to work with batches. Moreover our algorithms exploit information about the setup times between the consecutive batches, which further improves domain filtering. We have also designed a new filtering algorithm called not-before/not-after. Because the above filtering algorithms can remove different inconsistencies, we propose to integrate them into a single algorithm where the filtering is repeated until any domain is changed. Note that the underlying solving system calls this global filtering algorithm whenever a lower bound or an upper bound of the domain of any involved variable is changed by another constraint. It means that our filtering algorithm can be naturally integrated into a constraint-based scheduler. In particular, it is possible

to impose additional constraints over the problem variables (e.g. the direct relations between the activities) or to model more resources using several instances of the same filtering algorithm.

The paper is organized as follows. We first introduce the basic notions and define the functions that are pre-computed once at start and then used in our filtering algorithms. The main part of the paper describes the three filtering algorithms. We conclude with some experimental results.

Batch Processing – Basic Notions

Our definition of batch processing is close to p-batching in traditional scheduling (Brucker 2001), i.e. if there are two overlapping activities in the resource then these activities must start at the same time and they must complete at the same time. The activities processed together form a batch.

We can restrict the number of activities processed together in a single batch using capacity and compatibility constraints (Figure 1). Let T be a set of all activities that can be processed by the resource. Each activity $i \in T$ has assigned two attributes: c_i indicating the capacity consumed by the activity and f_i indicating the type of the activity (family). The compatibility constraint requires that only the activities of the same family are processed in a single batch. The capacity constraint says that the sum of the capacities of the activities in a single batch must not exceed the capacity C of the resource (renewable resources are assumed only, i.e. the capacity is consumed only when the activity runs). For simplicity reasons, we assume that the capacity C of the resource is a constant number.



Figure 1: Two tasks i and j of the same family ($f_i = f_j$) can be processed simultaneously.

Because all the activities processed in a single batch start and complete at the same time, they have the same duration. Thus we can assign the duration attribute to the family rather than to a particular activity. Formally, let F denotes the set of all the families in the resource, i.e., $F = \{f_i, i \in T\}$. Then p_f denotes the duration (processing time) of every activity of family f .

When we have a pair of consecutive batches processed by the resource, a special setup time have to be inserted between these batches. If this time depends on both batches then we are speaking about a sequence dependent setup time. If the setup time depends on one batch only then this time can be included in the processing time of this batch. Formally, we denote s_{fg} a setup time between the batches of family f and family g . No setup time is assumed between the batches of the same family, i.e.:

$$\forall f \in F : s_{ff} = 0 \quad (1)$$

or this setup time can be included in p_f . Like (Brucker & Thiele 1996) we assume that the setup time satisfies a triangle inequality:

$$\forall f, g, h \in F : s_{fh} \leq s_{fg} + s_{gh} \quad (2)$$

We are aware about the resources without this property, then a weaker version of our filtering algorithms must be used. Note finally that the notations of processing times for families can be extended to activities as well:

$$p_i = p_{f_i}$$

The scheduling task is to find out when the activity is processed by the resource, i.e. to determine its start time and its completion time respecting the above constraints. Initially, a release time r_i (the earliest start time) and a due time d_i (the latest completion time) is assigned to each activity $i \in T$. The release and due times defines a time window when the activity can be processed. The goal of the filtering algorithm is to shrink the time window by removing as much as possible infeasible times. More precisely, the release time is increased and the due time is decreased during filtering.

Let n denotes the number of activities, i.e., $n = |T|$ and k be the number of families, $k = |F|$. We assume that k is much smaller than n . Our filtering algorithms are polynomial in k and n , as we show later their time complexity is $O(kn^2)$. But the filtering algorithms require some preprocessing when a collated information is computed (see next section). Time complexity of preprocessing is $O(k^2 2^k)$ that is OK, if k is not a large number. We can assume that k is not large as it denotes the number of activity families.

Consolidated Setup Time

In the previous section, the attributes of the activity families have been introduced. We can now extend these notions into the sets of families. Opposite to (Brucker & Thiele 1996) we propose to compute these consolidated attributes once before the scheduling starts. We can identify a set of families using a standard bitmap, i.e. each set can be represented as a number. Thus the values of the consolidated attributes can be saved in an array with the access time $O(1)$.

We have defined a setup time between a pair of activity families. It is useful to know what is the minimal setup time when a set of activities is processed. In particular, let $\phi \subseteq F$ be a set of activity families, then $s(\phi)$ denotes the minimal setup time used when the activities of types in ϕ are processed. Similarly, $s(f, \phi)$ denotes the minimal setup time when processing starts with some activity of the type $f \in \phi$ and $s(\phi, f)$ is a minimal setup time when processing completes with an activity of the type $f \in \phi$. If we assume that the number of activity families is not very large, we can compute the values $s(\phi)$, $s(f, \phi)$, and $s(\phi, f)$ for every set ϕ of families and every family f in advance. As we show now, the time complexity of such algorithm is $O(k^2 2^k)$.

Visibly:

$$\forall \phi \subseteq F : s(\phi) = \min\{s(f, \phi), f \in \phi\} \quad (3)$$

Thus, it is enough to compute the functions $s(f, \phi)$ and $s(\phi, f)$; the function $s(\phi)$ can then be computed using the

formula (3) in time $O(k2^k)$. We show now how to compute the function $s(f, \phi)$ inductively by the size of the set ϕ . The function $s(\phi, f)$ can be computed in a similar way.

For the sets ϕ with just one family, there are no setups used according to (1):

$$\forall f \in F : s(f, \{f\}) = 0$$

We can compute the value $s(f, \{f\} \cup \phi)$ from the value $s(g, \phi)$ using the following formula (note, that the triangle inequality (2) is required here):

$$\forall \phi \subset F, \forall f \in (F \setminus \phi) : \\ s(f, \{f\} \cup \phi) = \min\{s_{fg} + s(g, \phi), g \in \phi\}$$

The time complexity of this computation is $O(k^2 2^k)$.

Consolidated Processing Time

Similarly to the minimal setup time when processing the set of activities, we can define the minimal processing time for a given set of activities. Now the capacity of the activities and the capacity of the resource are assumed.

Let $\Omega \subseteq T$ be a set of activities, then $c(\Omega, f)$ denotes the total capacity of the activities from Ω that have the family f :

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

Let $u(\Omega, f)$ denotes the minimal time to process all the activities of family f from Ω . This value is computed as a multiplication of the minimal number of batches and the processing time of the family f :

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

Let F_Ω be the set of all the activity families in the set Ω , i.e. $F_\Omega = \{f_i, i \in \Omega\}$. Now we can define the total pure processing time $u(\Omega)$ for the set of activities Ω when no setups are involved:

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

Finally, we define the consolidated processing time $p(\Omega)$ for the set of activities Ω which consists of the pure processing time and the setup time for Ω .

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

Like the setup times, consolidated processing time can be specified when processing starts or completes with an activity of a particular family:

$$p(j, \Omega) = s(f_j, F_\Omega) + u(\Omega) \\ p(\Omega, j) = s(F_\Omega, f_j) + u(\Omega)$$

Let d_Ω be the maximal due time among the activities in the set Ω and r_Ω be the minimal release time of the activities in the set Ω :

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

Filtering Rules

In this section we show a modification of the edge-finding rules from (Martin & Shmoys 1996) to work with batch processing and sequence dependent setup times. We also propose the rules not-before/not-after that further increase the filtering power of edge-finding. The rules not-first/not-last will be described in a separate section.

Integrity Rule

The feasible schedule exists only if all the activities in every set $\Omega \subseteq T$ can be processed within the time interval $\langle r_\Omega, d_\Omega \rangle$. The following rule deduces that a feasible schedule does not exist because there is not enough time to process the activities from Ω :

$$\forall \Omega \subseteq T : d_\Omega - r_\Omega < p(\Omega) \Rightarrow \text{fail} \quad (4)$$

Not-Before/Not-After Rules

Consider an arbitrary set $\Omega \subset T$ and an activity $i \notin \Omega$. If we schedule the activity i before the activities Ω , then processing of the set Ω can start at first in the time $r_i + p_i$. Then processing of activities in Ω needs time $u(\Omega) + s(f_i, F_\Omega \cup \{f_i\})$ because the resource is already adjusted for processing of f_i (this is the family of the activity i that is processed before Ω). If such a schedule is not possible (i.e. processing of Ω ends after d_Ω) then the activity i cannot be scheduled before Ω :

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega) : \\ r_i + p_i + u(\Omega) + s(f_i, F_\Omega \cup \{f_i\}) > d_\Omega \Rightarrow i \not\prec \Omega \quad (5)$$

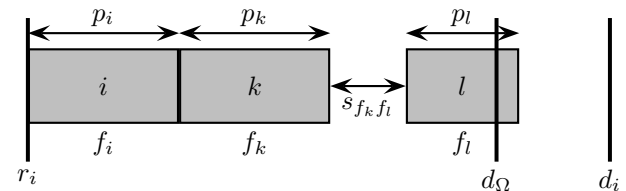


Figure 2: Example of not-before condition for the task i and set $\Omega = \{k, l\}$, when $f_i = f_k$.

When the activity i cannot be processed before the activities in Ω then processing of the activity i can start at first together with the first activity from Ω . Thus we get the following *not-before* rule for modification of the earliest start time of the activity i :

$$i \not\prec \Omega \Rightarrow r_i \geq r_\Omega \quad (6)$$

A symmetric rule *not-after* is defined in the following way:

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega) : \\ d_i - p_i - u(\Omega) - s(F_\Omega \cup \{f_i\}, f_i) < r_\Omega \Rightarrow \Omega \not\prec i \quad (7)$$

$$\Omega \not\prec i \Rightarrow d_i \leq d_\Omega \quad (8)$$

Edge-Finding Rules

Like the not-before/not after rules, the edge finding rule is trying to find a relative position of the activity i in respect to the set of activities Ω . Now we are asking whether the activity i can be processed together with the activities in Ω . If the answer is no then i must be processed either before or after all the activities in Ω . Consider again an arbitrary set $\Omega \subset T$ and an activity $i \notin \Omega$. If there is not enough time for processing activities $\Omega \cup \{i\}$ in the interval $\langle r_\Omega, d_\Omega \rangle$, then the activity i has to be scheduled before or after all the activities from Ω :

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega) : \\ d_\Omega - r_\Omega < p(\Omega \cup \{i\}) \Rightarrow (\Omega \ll i \text{ or } i \ll \Omega) \quad (9)$$

Now we need to decide if the activity i has to be scheduled before Ω or after it. For that decision we can use the not-before and not-after rules. If one of these rules ((5) or (7)) holds then we can deduce the relative position of i in respect to Ω and we can change r_i or d_i accordingly:

$$\Omega \ll i \Rightarrow \quad (10)$$

$$r_i \geq \max\{r_{\Omega'} + u(\Omega') + s(F_{\Omega'} \cup \{f_i\}, f_i), \Omega' \subseteq \Omega\} \\ i \ll \Omega \Rightarrow \quad (11)$$

$$d_i \leq \min\{d_{\Omega'} - u(\Omega') - s(f_i, F_{\Omega'} \cup \{f_i\}), \Omega' \subseteq \Omega\}$$

How to choose Ω efficiently?

The above rules can be applied to all the sets $\Omega \subseteq T$ to achieve the strongest domain filtering. This leads to a filtering algorithm exponential in the number of activities. Fortunately, we can apply the rules to a polynomial number of selected sets $\Omega \subseteq T$ while keeping the same filtering power. We show now how these sets can be designed.

Consider an arbitrary set Ω and a set Ψ defined in the following way:

$$\Psi = \{k, k \in T \ \& \ r_k \geq r_\Omega \ \& \ d_k \leq d_\Omega\}$$

Caseau & Laburthe call such a set a *tasks interval* (1994, 1995, 1996). The set Ψ has the following properties:

$$r_\Psi = r_\Omega \\ d_\Psi = d_\Omega \\ \Psi \supseteq \Omega \\ u(\Psi) \geq u(\Omega) \\ F_\Psi \supseteq F_\Omega$$

Therefore if we use Ψ instead of Ω in the above rules then we get the same or even better results. Thus we can apply the rules only to the tasks intervals in the form:

$$\Psi = [m, n] = \{k, k \in T \ \& \ r_k \geq r_m \ \& \ d_k \leq d_n\}$$

where m, n are such activities that $r_m \leq r_n$ and $d_m \leq d_n$. Note that the number of tasks intervals is quadratic in the number of activities.

An Algorithm for the Integrity Rule

Now we present an algorithm that checks the integrity rule for every tasks interval. Time complexity of this algorithm is $O(n^2)$. The algorithm assumes that all the activities are sorted in the decreasing order of r_i what could be done in time $O(n * \log(n))$. For each activity j the algorithm constructs incrementally all valid tasks intervals $[i, j]$ and for each such interval it checks the integrity rule (4):

```

for  $j \in T$  do begin
   $\Omega := \emptyset$ ;
  for  $i \in T$  in the decreasing order of  $r_i$  do begin
    if  $d_i > d_j$  then
      //  $[i, j]$  is not a tasks interval
      continue;
    // expand  $\Omega$  to a tasks interval  $[i, j]$ 
     $\Omega := \Omega \cup \{i\}$ ;
    if  $d_j - r_i < p(\Omega)$  then fail ;
  end;
end;

```

An Algorithm Not-Before/Not-After

The algorithms for the rules not-before and not-after are symmetrical so we describe the algorithm only for the rule not-before. Again, the algorithm explores all valid tasks intervals $[j, k]$. Nevertheless, we do not need to check the condition of the not-before rule (5) for each tasks interval separately. The following theorem gives a hint how to do it more efficiently.

Theorem 1 *Let i be an activity of the family g and j be another arbitrary activity. The rules not-before (i.e. (5) and (6)) propose change of the value r_i using $r_i \geq r_j$ if and only if:*

$$r_i > \min\{d_\Omega - s(g, F_\Omega \cup \{g\}) - u(\Omega) - p_g, \\ \Omega = [j, k], k \in T\} \quad (12)$$

Proof: The inequality (12) holds if and only if there exists a tasks interval $[j, k]$ such that (5) holds. As we have shown above, we can work with the tasks intervals only. In particular, if the condition in (5) is true for some set Ω then we can find a tasks interval $[j, k]$ such that the condition holds as well. \square

Notice that the right side of the inequality (12) is independent of a particular activity i . It depends on the family g of the activity i only so we can compute the value of the right side just once and then use the result for all the activities of the family g . The following algorithm does exactly this:

```

for  $g \in F$  do begin
  for  $j \in T$  do begin
     $m := \infty$ ;
     $\Omega := \emptyset$ ;
    for  $k \in T$  in the increasing order of  $d_k$  do begin
      if  $r_k < r_j$  then
        //  $[j, k]$  is not a tasks interval
        continue;
       $\Omega := \Omega \cup \{k\}$ ;
       $m := \min\{d_\Omega - s(g, F_\Omega \cup \{g\}) - u(\Omega) - p_g, m\}$ ;
    end;
  end;

```

```

for  $i \in T, f_i = g$  do
  if  $r_i > m$  then
     $r_i := \max(r_i, r_j)$ ;
end;
end;

```

Time complexity of this algorithm is $O(kn^2)$.

An Algorithm Edge-Finding

In this section we present an algorithm only for the case $\Omega \ll i$ of the edge-finding rule. The algorithm for $i \ll \Omega$ can be designed in a symmetrical way.

First, let us analyze which pairs of the activity and the tasks interval should be checked to change the earliest start time for the activity. Let us choose an activity j and a family g . We now deduce all the changes for the activities with the family g resulting from the application of the edge-finding rule to all the tasks intervals $[i, j]$.

Let $\Omega_0 \subsetneq \Omega_1 \subsetneq \dots \subsetneq \Omega_x$ be a sequence of all the tasks intervals $[i, j]$ such that $d_{\Omega_0} = d_{\Omega_1} = \dots = d_{\Omega_x} = d_j$. Let t_0, t_1, \dots, t_y denotes the activities of the family g sorted in the increasing order of required capacity c_i . Now consider an application of the edge-finding rule to the pair Ω_x, t_y . One of the four cases must happen:

1. $d_{t_y} \leq d_{\Omega_x}$. In this case we will show that if the conditions of the edge-finding rules (5) and (9) hold then the symmetrical algorithm edge-finding for $i \ll \Omega$ together with the integrity rule deduces fail. Therefore we can ignore this case.

Similarly the symmetrical algorithm for $i \ll \Omega$ ignores the case when $r_{t_y} \geq r_{\Omega_x}$. So the pair Ω_x and t_y is totally ignored by the whole edge-finding algorithm when $d_{t_y} \leq d_{\Omega_x}$ and $r_{t_y} \geq r_{\Omega_x}$ meaning that in this case we cannot apply the edge-finding rules at all because $t_y \in \Omega_x$.

Let us now assume that for t_y and Ω_x the inequalities in the conditions of both rules (5) and (9) hold. $d_{t_y} \leq d_{\Omega_x}$ and (9) implies:

$$d_{t_y} - r_{\Omega_x} \leq d_{\Omega_x} - r_{\Omega_x} < p(\Omega_x \cup \{t_y\})$$

It is obvious that:

$$p(\Omega_x \cup \{t_y\}) \leq u(\Omega_x) + p_g + s(F_{\Omega_x} \cup \{g\}, g)$$

Therefore for t_y and Ω_x the condition of the rule (7) holds and together with (9) we get $i \ll \Omega$. So we can change d_{t_y} using (11):

$$d_{t_y} \leq d_{\Omega_x} - u(\Omega_x) - s(g, F_{\Omega_x} \cup \{g\})$$

The inequality in the condition of the rule (5) for t_y and Ω_x still holds and thus:

$$r_{t_y} > d_{\Omega_x} - p_g - u(\Omega_x) - s(g, F_{\Omega_x} \cup \{g\})$$

Now consider the set $\Psi = \{t_y\}$:

$$\begin{aligned}
d_{\Psi} - r_{\Psi} &= d_{t_y} - r_{t_y} \leq \\
&\leq [d_{\Omega_x} - u(\Omega_x) - s(g, F_{\Omega_x} \cup \{g\})] - r_{t_y} < \\
&< [d_{\Omega_x} - u(\Omega_x) - s(g, F_{\Omega_x} \cup \{g\})] - \\
&\quad [d_{\Omega_x} - p_g - u(\Omega_x) - s(g, F_{\Omega_x} \cup \{g\})] = \\
&= p_g = p_{\Psi}
\end{aligned}$$

Hence $d_{\Psi} - r_{\Psi} < p_{\Psi}$ and the integrity rule (4) deduces fail.

We have shown that in the case $d_{t_y} \leq d_{\Omega_x}$ the rules (5) and (9) can be ignored because if the conditions of these rules hold then the integrity rule deduces fail. And because $d_{\Omega_0} = d_{\Omega_1} = \dots = d_{\Omega_x}$ these rules can be ignored also for the activity t_y and any set $\Omega_0, \Omega_1, \dots, \Omega_x$. Hence we can remove t_y from the sequence t_0, t_1, \dots, t_y .

In the following three cases we expect that $d_{t_y} > d_{\Omega_x}$ and so $t_y \notin \Omega_x$.

2. **The set Ω_x and the activity t_y do not satisfy the condition in (9)** and so the activity t_y can be processed together with Ω_x . Then any activity t_0, t_1, \dots, t_{y-1} can be processed together with Ω_x as well because it requires the same or less capacity than the activity t_y . Hence we can remove the set Ω_x from the sequence $\Omega_0, \Omega_1, \dots, \Omega_x$.
3. **The set Ω_x and the activity t_y do not satisfy the condition (5)** and so the activity t_y can be processed before the Ω_x . Then the activity t_y can be processed before any set $\Omega_0, \Omega_1, \dots, \Omega_{x-1}$. Hence we can remove the activity t_y from the sequence t_0, t_1, \dots, t_y .
4. **The set Ω_x and the activity t_y satisfy both conditions (5) and (9)**. Then we can change value r_{t_y} using the rule (10). This new value cannot be further increased using any set $\Omega_0, \Omega_1, \dots, \Omega_{x-1}$ because of the form of the rule (10). Hence we can remove t_y from the sequence t_0, t_1, \dots, t_y .

The above analysis explains what pairs of activity and tasks interval needs to be checked. We have shown that only the case 4 above contributes to change of the value r_{t_y} using the rule (10). When we want to change the value r_{t_y} according to the rule (10) then we need to know the following value z_x :

$$z_x = \max\{r_{\Omega'} + u(\Omega') + s(F_{\Omega'} \cup \{g\}, g), \Omega' \subseteq \Omega_x\}$$

Now we show how to compute the value z_x efficiently.

Consider an arbitrary set $\Omega' \subseteq \Omega_x$ and the set $\Phi = \{k, k \in \Omega_x \ \& \ r_k \geq r_{\Omega'}\}$. It is obvious that:

$$r_{\Omega'} + u(\Omega') + s(F_{\Omega'} \cup \{g\}, g) \leq r_{\Phi} + u(\Phi) + s(F_{\Phi} \cup \{g\}, g)$$

So we can choose only the sets Ω' in the form of a tasks interval $[i, j]$ such that $d_j = d_{\Omega}$. These tasks intervals are exactly the sets $\Omega_0, \Omega_1, \dots, \Omega_x$. Hence:

$$\begin{aligned}
z_x &= \max\{r_{\Omega'} + u(\Omega') + s(F_{\Omega'} \cup \{g\}, g), \\
&\quad \Omega' \in \{\Omega_0, \Omega_1, \dots, \Omega_x\}\}
\end{aligned}$$

Therefore we can compute z_x inductively using the formula:

$$z_x = \max\{z_{x-1}, r_{\Omega_x} + u(\Omega_x) + s(F_{\Omega_x} \cup \{g\}, g)\}$$

Using the above analysis we can now design an algorithm implementing the combination of the edge-finding and not-before rules. Time complexity of this algorithm is $O(kn^2)$. As usual, the algorithm combining the rules edge-finding and not-after can be designed in a symmetrical way.

The following algorithm assumes that the activities are sorted decreasingly by values of r_i and also by values of c_i , i.e. two sorted lists of activities are prepared in advance. The algorithm generates first a sequence of the tasks intervals $[i, j]$. Then it compares these tasks intervals with all the activities of a given family g using the above four cases analysis until either the list of tasks intervals or the list of activities is completely explored.

```

for  $g \in F$  do begin
  for  $j \in T$  do begin
     $\Omega_{-1} := \emptyset$ ;
     $z_{-1} := -\infty$ ;
     $x := 0$ ;
    for  $i \in T$  in the decreasing order of  $r_i$  do begin
      if  $d_i > d_j$  then
        //  $[i, j]$  is not a tasks interval
        continue;
       $\Omega_x := \Omega_x \cup \{i\}$ ;
       $z_x := \max(z_{x-1}, r_i + u(\Omega_x) + s(F_{\Omega_x} \cup \{g\}))$ ;
       $x := x + 1$ ;
    end;
     $y :=$  an activity with the family  $g$  with the greatest  $c_y$ ;
    while ( $y \geq 0$  and  $x \geq 0$ ) do begin
      if  $d_y \leq d_j$  then
        // case 1
         $y :=$  next activity in the sequence or -1;
        continue;
      end;
      if  $d_j - r_{\Omega_x} \geq p(\Omega_x \cup \{y\})$  then
        // case 2
         $x := x - 1$ ;
        continue;
      end;
      if  $d_j - r_y \geq s(g, F_{\Omega_x} \cup \{g\}) + u(\Omega_x) + p_g$  then
        // case 3
         $y :=$  next activity in the sequence or -1;
        continue;
      end;
      // case 4
       $r_y := \max(r_y, z_x)$ ;
       $y :=$  next activity in the sequence or -1;
    end;
  end;
end;

```

Not-First/Not-Last

In this section we present an extension of another filtering technique called not-first/not-last that was proposed by (Baptiste & Le Pape 1996). Note that this extension is different from what we call not-before/not-after as explained below.

The Rules

Unlike not-before/not-after we now consider an activity i and a set $\Omega \subset T$ such that i cannot start neither before Ω nor together with the first activity in Ω . We denote such

property $i \not\prec \Omega$, similarly $\Omega \not\prec i$:

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega) : \\ d_\Omega - r_i < p(i, \Omega \cup \{i\}) \Rightarrow i \not\prec \Omega \quad (13)$$

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega) : \\ d_i - r_\Omega < p(\Omega \cup \{i\}, i) \Rightarrow \Omega \not\prec i \quad (14)$$

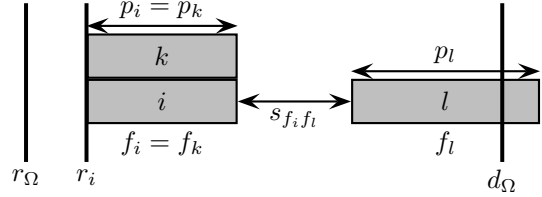


Figure 3: Example of not-first condition for the task i and set $\Omega = \{k, l\}$, when $f_i = f_k$.

If $i \not\prec \Omega$ then i can start at first when the first activity of Ω finishes, i.e., in the second batch of Ω , and similarly for $\Omega \not\prec i$:

$$i \not\prec \Omega \Rightarrow r_i \geq \min\{r_j + p_j + s_{f_j f_i}, j \in \Omega\} \quad (15)$$

$$\Omega \not\prec i \Rightarrow d_i \leq \max\{d_j - p_j - s_{f_i f_j}, j \in \Omega\} \quad (16)$$

Unfortunately, we were not able to make up an algorithm for the rules (13) and (14) with reasonable time complexity. Therefore we propose a weaker version of these rules which allow us to design an algorithm with the time complexity $O(kn^2)$.

Let $f \in F$ be a family then c_f is a minimal capacity among the activities from the family f , $t(f, \Omega)$ is a minimal value of $p(i, \Omega \cup \{i\})$ among all the activities i of the family f , and symmetrically $t(\Omega, f)$. Formally c_f , $t(f, \Omega)$, and $t(\Omega, f)$ denote:

$$c_f = \min\{c_i, i \in T \ \& \ f_i = f\}$$

$$t(f, \Omega) = s(f, F_\Omega \cup \{f\}) + u(\Omega) -$$

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

$$t(\Omega, f) = s(F_\Omega \cup \{f\}, f) + u(\Omega) -$$

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

Then the weaker replacement of the rules (13) and (14) is:

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega) : \\ d_\Omega - r_i < t(f_i, \Omega) \Rightarrow i \not\prec \Omega \quad (17)$$

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega) : \\ d_i - r_\Omega < t(\Omega, f_i) \Rightarrow \Omega \not\prec i \quad (18)$$

An Algorithm

We describe an algorithm only for the rules not-first i.e. (13), (15) and (17), the algorithm for the rules not-last is symmetrical.

Hereafter we assume that the activities are ordered increasingly by the values d_i , i.e. $i \leq j \Leftrightarrow d_i \leq d_j$. Let us choose three arbitrary activities i, j, k such that:

$$j \leq k \quad (19)$$

$$r_j + p_j + s_{f_j f_i} \leq r_k + p_k + s_{f_k f_i} \quad (20)$$

i.e., if $i \notin \{j, k\}$ then j proposes an earlier start time for i . Let $\Omega(fjk)$ and $\Omega(ijk)$ denote the sets:

$$\begin{aligned}\Omega(fjk) &= \{m, m \in T \ \& \ m \leq k \ \& \\ & \quad r_m + p_{f_m} + s_{f_m f} \geq r_j + p_{f_j} + s_{f_j f}\} \\ \Omega(ijk) &= \Omega(fijk) \setminus \{i\}\end{aligned}$$

The set $\Omega(ijk)$ contains all the activities that do not finish after the activity k ($m \leq k$, i.e., $d_m \leq d_k$) and that do not propose an earlier start time for activities of the family f than the activity j . When j and k do not satisfy the condition (19) or (20) then let $\Omega(fjk) = \emptyset$ and $d_{\Omega(fjk)} = \infty$.

We can now show that the sets $\Omega(ijk)$ play the same role for the not-first rule like the tasks intervals play for the edge-finding rule. It means that we can use only the sets $\Omega(ijk)$ in the rules (13), (15) and (17) instead of arbitrary sets Ω . The following theorem justifies this claim.

Theorem 2 *If the rules (13 or 17) and (15) for the activity i and the set Ω change the value r_i to $r_j + p_j + s_{f_j f_i}$ then there exists an activity $k \in T$ such that $k \neq i$, $k \geq j$ and the same rules for the activity i and the set $\Omega(ijk)$ also change the value r_i to $r_j + p_j + s_{f_j f_i}$.*

Proof: Let $k = \max\{m, m \in \Omega\}$. It is obvious that $j \leq k$ because $j \in \Omega$. The choice of k implies $\Omega \subseteq \Omega(ijk)$ and $d_{\Omega} = d_{\Omega(ijk)}$. Because (13 or 17) holds for Ω the same rule holds even for $\Omega(ijk)$ and so the rule (15) deduces $r_i \geq r_j + p_j + s_{f_j f_i}$. \square

We will not use the sets $\Omega(ijk)$ directly in the filtering algorithm but via the following theorem. First, let us establish a new notation:

$$\begin{aligned}\gamma_{j,k}(f) &= \min\{d_{\Omega(fjl)} - s(f, F_{\Omega(fjl)}) - u(\Omega(fjl)), \\ & \quad l \in \{k, k+1, \dots, n\}\} \\ \delta_{j,k}(f) &= \min\{d_{\Omega(fjl)} - t(f, \Omega(fjl)), l \in \{1, 2, \dots, k\}\}\end{aligned}$$

Briefly speaking, $\gamma_{j,k}(f)$ is a minimal value of $d_{\Omega(fjk)} - p(f, \Omega(fjk))$ which can be used in the condition of the rule (13) for activities of the family f . Similarly, $\delta_{j,k}(f)$ can be used to check the condition in (17). Note also that for a given activity j and a family f we can compute the values $\gamma_{j,k}(f)$ and $\delta_{j,k}(f)$ in the time $O(n)$.

Theorem 3 *If arbitrary activities i, j satisfy at least one of the two conditions:*

1. $r_i + p_i < r_j + p_j + s_{f_j f_i}$ and $\delta_{j,n}(f_i) < r_i$.
2. $r_i + p_i \geq r_j + p_j + s_{f_j f_i}$ and $(\delta_{j,i-1}(f_i) < r_i$ or $\gamma_{j,i}(f_i) < r_i)$.

then the rules (13) and (15) allow us to change the value r_i using $r_i \geq r_j + p_j + s_{f_j f_i}$.

If i and j do not satisfy neither 1. nor 2. then the change $r_i \geq r_j + p_j + s_{f_j f_i}$ can't be achieved using the weaker rule (17).

Proof: We distinguish between two cases:

1. $r_i + p_i < r_j + p_j + s_{f_j f_i}$. Then for an arbitrary activity k we get $i \notin \Omega(fijk)$ and so $\Omega(ijk) = \Omega(fijk)$. According to Theorem 2 the rule (17) allows us to change

r_i using $r_i \geq r_j + p_j + s_{f_j f_i}$ if and only if there exists such an activity l that:

$$d_{\Omega(f_i j l)} - t(f, \Omega(f_i j l)) < r_i$$

And this activity exists iff:

$$\min\{d_{\Omega(f_i j l)} - t(f, \Omega(f_i j l)), l \in T\} < r_i$$

Which holds iff $\delta_{j,n}(f_i) < r_i$.

2. $r_i + p_i \geq r_j + p_j + s_{f_j f_i}$. We distinguish between other two cases:

- $i \notin \Omega(f_i j l)$ that is equivalent to $i > l$. Then $\Omega(ijl) = \Omega(f_i j l)$ and according to Theorem 2 the rule (17) can deduce $r_i \geq r_j + p_j + s_{f_j f_i}$ if and only if there exists such an activity l that:

$$d_{\Omega(ijl)} - t(f_i, \Omega(ijl)) < r_i$$

Such l exists iff:

$$\min\{d_{\Omega(f_i j l)} - t(f, \Omega(f_i j l)), l \in \{1, \dots, i-1\}\} < r_i$$

That is exactly:

$$\delta_{j,i-1}(f_i) < r_i$$

- $i \in \Omega(f_i j l)$ so $l > i$ (in fact $l \geq i$ but according to theorem 2 $l \neq i$ and thus $l > i$). This time we use the stronger rule (13). According to Theorem 2 the rule (13) can deduce the change of r_i iff there exists such activity l that:

$$d_{\Omega(ijl)} - p(i, \Omega(ijl) \cup \{i\}) < r_i$$

Because $i \in \Omega(f_i j l)$ thus $\Omega(ijl) = \Omega(f_i j l)$. And so such activity l exists if and only if:

$$\min\{d_{\Omega(f_i j l)} - p(i, \Omega(f_i j l)), l \in \{i, \dots, n\}\} < r_i$$

And this holds iff $\gamma_{j,i}(f_i) < r_i$. \square

The above theorem provides instructions how to realize the not-first rule. First, we compute the values $\gamma_{j,k}(f)$ and $\delta_{j,k}(f)$ and then we use the conditions from Theorem 2 to find out when the earliest start time of some activity i can be moved forward. The time complexity of such algorithm is $O(kn^2)$.

for $f \in F$ do begin

for $j \in T$ do begin

compute $\gamma_{j,1}(f), \gamma_{j,2}(f), \dots, \gamma_{j,n}(f)$;

compute $\delta_{j,1}(f), \delta_{j,2}(f), \dots, \delta_{j,n}(f)$;

for $i \in T, f_i = f$ do

if $r_i + p_i < r_j + p_j + s_{f_j f}$ then begin

if $\delta_{j,n}(f_i) < r_i$ then

$r_i := \max(r_i, r_j + p_j + s_{f_j f});$

end else begin

if $\delta_{j,i-1}(f_i) < r_i$ or $\gamma_{j,i}(f_i) < r_i$ then

$r_i := \max(r_i, r_j + p_j + s_{f_j f});$

end;

end;

end;

Experimental Results

As far as we know there is no standard set of benchmark tests for the problem of batch processing with sequence dependent setup times. Therefore we designed own benchmark set to test and compare our filtering techniques. The test data has been generated in the following way: a feasible schedule for a given number of activities has been generated and then the time windows for the activities has been randomly extended. In particular, if the processing of the activity i starts at time t_i in the schedule then we set values r_i and d_i in the following way:

$$r_i = t_i - \text{rand}(m)$$
$$d_i = t_i + \text{rand}(m) + p_i$$

The function $\text{rand}(m)$ generates random integer numbers in the range $0 \dots (m - 1)$. We use only small m to keep the problem tight. Generated problems can be found at (Vilím & Barták 2002a).

The individual filtering techniques have been combined into a single filtering algorithm. In this algorithm, the strongest technique – edge-finding is used in the inner loop to call them more frequently while not-first/not-last is in the outer loop to be called less frequently. The idea is to call these filtering techniques until any domain changes:

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

The above structure of the filtering algorithm has been deduced primarily from the experimental results. We discuss this structure in more details in Conclusions. The tables 1 and 2 show the computational results measured on Intel Pentium Celeron 375MHz.

Conclusions

Filtering techniques play a significant role in pruning the search space. In scheduling, some problems were not schedulable (in a reasonable time) without using a filtering technique like edge-finding or not-first/not-last. In this paper we propose an extension of these algorithms to batch processing with sequence dependent setup times. We also add a new technique not-before/not-after that further improves the filtering.

The above techniques filter different inconsistencies so it is natural to combine them into a single filtering algorithm. In this algorithm, these techniques are repeatedly applied until any domain changes. The experimental results showed that edge-finding and not-first/not-last techniques deduce the largest number of changes among the presented filtering techniques and thus in our joined algorithm these techniques are repeated more frequently than the technique

not-before/not-after. Such structure of the global filtering algorithm does not change the filtering power, i.e. the same inconsistencies are removed independently on the ordering of calls to a particular technique. However, this structure improves significantly the runtime because the idle calls that deduce no change are suppressed.

The experimental results approved the significance of edge-finding technique. If the not-before/not-after techniques do not reduce the search space (Table 2) then using them slows down the computation for 20% on average due to overhead. However when not-before/not-after proposes some domain reduction then the computation time can be significantly shorter. We expect that in real-life not-before/not-after rules pay off because reductions proposed by them make cause further reductions from additional constraints.

Together with the filtering techniques described in this paper, we have developed one more filtering algorithm for batch processing with sequence dependent setup times called *sequence composition*. However, the experimental results show that the contribution of this algorithm to filtering is not as significant as in the case of above algorithms. Due to space restrictions we omit the description of this filtering technique here, it can be found in (Vilím & Barták 2002b).

Acknowledgements

The research is supported by the Grant Agency of the Czech Republic under the contract no. 201/01/0942.

| problem | n | k | solutions | backtracks | time | without not-before/not-after | |
|---------|----|---|-----------|------------|---------|------------------------------|---------|
| | | | | | | backtracks | time |
| a | 30 | 3 | 88 | 12 | 2.22s | 13 | 1.94s |
| b | 25 | 5 | 56251 | 5016 | 25m 30s | 104880 | 44m 38s |
| e | 20 | 2 | 28 | 0 | 0.20s | 16 | 0.25s |
| g | 30 | 3 | 1690 | 77 | 37.08s | 92 | 32.71s |
| h | 75 | 5 | 12 | 2 | 2.90s | 8 | 3.26s |
| i | 50 | 5 | 48 | 44 | 7.59s | 220 | 18.17s |
| j | 50 | 5 | 10 | 4 | 1.34s | 21 | 1.84s |
| k | 50 | 7 | 9 | 0 | 1.33s | 24 | 2.93s |
| l | 50 | 5 | 4 | 0 | 0.51s | 3 | 0.51s |
| n | 30 | 5 | 39 | 13 | 1.78s | 37 | 1.87s |
| p | 30 | 3 | 270 | 24 | 6.05s | 222 | 7.32s |
| r | 50 | 7 | 324 | 0 | 44.94s | 804 | 1m 28s |
| z | 50 | 4 | 24 | 7 | 2.14s | 9 | 1.92s |

Table 1: Cases when not-before/not-after reduces number of backtracks.

| problem | n | k | solutions | backtracks | time | without not-before/not-after | |
|---------|-----|---|-----------|------------|--------|------------------------------|--------|
| | | | | | | backtracks | time |
| c | 25 | 5 | 72 | 0 | 1.78s | 0 | 1.45s |
| d | 40 | 6 | 12 | 1 | 1.02s | 1 | 0.81s |
| f | 50 | 6 | 6 | 2 | 0.70s | 2 | 0.59s |
| m | 50 | 3 | 3 | 3 | 0.35s | 3 | 0.29s |
| o | 50 | 5 | 32 | 8 | 3.60s | 8 | 2.65s |
| q | 50 | 7 | 228 | 0 | 28.03s | 0 | 24.08s |
| s | 100 | 7 | 50 | 0 | 26.48s | 0 | 22.20s |
| t | 200 | 7 | 8 | 0 | 18.00s | 0 | 14.96s |
| v | 100 | 7 | 240 | 0 | 2m 6s | 0 | 1m 46s |
| w | 50 | 2 | 24 | 4 | 1.36s | 4 | 1.14s |
| x | 50 | 2 | 1368 | 384 | 1m 8s | 384 | 58.25s |

Table 2: Cases when not-before/not-after does not help.

References

- Baptiste, P., and Le Pape, C. 1996. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group*.
- Baptiste, P.; Le Pape, C.; and Nuijten, W. 1998. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. In *Technical Report 98/97*. Universit  de Technologie de Compigne.
- Brucker, P., and Thiele, O. 1996. A branch and bound method for the general shop problem with sequence dependent setup-times. In *OR Spectrum*, 145–161. Springer-Verlag.
- Brucker, P. 2001. *Scheduling Algorithms*. Springer-Verlag, 3rd edition.
- Carlier, J., and Pinson, E. 1989. An algorithm for solving the job-shop problem. *Management Science* 35(2):164–176.
- Caseau, Y., and Laburthe, F. 1994. Improved CLP Scheduling with Task Intervals. In van Hentenryck, P., ed., *Pro-*

ceedings of the 11th International Conference on Logic Programming, ICLP'94. The MIT press.

Caseau, Y., and Laburthe, F. 1995. Disjunctive scheduling with task intervals. In *Technical report, LIENS Technical Report 95-25*. Ecole Normale Suprieure Paris, Franc.

Caseau, Y., and Laburthe, F. 1996. Cumulative scheduling with task intervals. In *Joint International Conference and Symposium on Logic Programming*, 363–377.

Martin, P., and Shmoys, D. B. 1996. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem. In Cunningham, W. H.; McCormick, S. T.; and Queyranne, M., eds., *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization, IPCO'96*, 389–403.

Vil m, P., and Bart k, R. 2002a. A benchmark set for batch processing with sequence dependent setup times. <http://kti.mff.cuni.cz/~vilim/batch>.

Vil m, P., and Bart k, R. 2002b. A filtering algorithm sequence composition for batch processing with sequence dependent setup times. Technical Report 2002/1, Charles University, Faculty of Mathematics and Physics.