Filtering Algorithm Sequence Composition 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

May 28, 2002

Abstract

There exist three filtering algorithms for batch processing with sequence dependent setup times: edge-finding, not-first/not-last and not-before/not-after. In this paper we propose another filtering algorithm called *sequence composition*. This algorithm can be used together with the previous three for even better pruning of search space. However, experimental results show that sequence composition is slow and yield only a few domain reductions.

1 Introduction

In ours previous work (Vilím and Barták 2002b) we proposed some filtering algorithms for batch processing with sequence dependent setup times. In this paper we describe one more algorithm – sequence composition. Each of these algorithms find different sort of inconsistencies therefore they can be used together.

This technical report closely bear on the paper (Vilím and Barták 2002b), we will use the notation, definitions and functions from this paper without explaining them again.

Consider following problem consisting of four tasks $T = \{1, 2, 3, 4\}$, three families $F = \{1, 2, 3\}$ and machine with capacity $C = 2$:

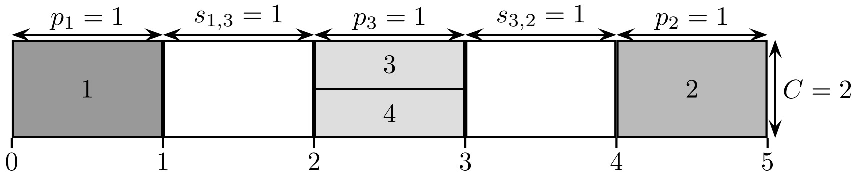
$$\forall f, g \in F: \; s_{fg} = \begin{cases} 0 \text{ when } f = g \\ 1 \text{ when } f \neq g \end{cases}$$

$$\forall f \in F: \; p_f = 1$$

$$\begin{aligned} f_1 = 1, \quad r_1 = 0, \quad d_1 = 5, \quad c_1 = 2 \\ f_2 = 2, \quad r_2 = 0, \quad d_2 = 5, \quad c_2 = 2 \\ f_3 = 3, \quad r_3 = 0, \quad d_3 = 3, \quad c_3 = 1 \\ f_4 = 3, \quad r_4 = 2, \quad d_4 = 5, \quad c_4 = 1 \end{aligned}$$

This problem has only two solutions. One of them is on the following picture, the second one is the same but the tasks 1 and 2 are swaped.

*Supported by the Grant Agency of the Czech Republic under the contract no. 201/01/0942



Edge-finding, not-first/not-last and not-before/not-after do not deduce any change of time bounds. However we see that r_3 and d_4 can be changed to: $r_3 = 2$, $d_4 = 3$. This the motivation for our algorithm.

The idea of sequence composition is to find out that two or more activities of the same type have to be processed without interruption by different family. From this knowledge we can find the change of the time bounds.

The paper is organized as follows: first we establish new precomputed function q, in the next section we show how to detect some sets of uninterruptible activities and propose the filtering rules. Finally we build up filtering algorithm and conclude with some experimental results.

2 Setup Time with Interruption

We need to recognize when the interruption between the activites with the same family is not possible. For that we need to now how long would be the setups when the interruption occurs. Therefore we define function q similar to function s (see Vilím and Barták 2002b). Value of $q(f, g, \phi)$ is the minimal setup time needed for processing the activities with families ϕ under the condition that the processing starts with an activity with family $f \in \phi$ and the processing of the the family $g \in \phi$ is interrupted at least once by an activity with the type from $\phi \setminus \{g\}$. The values of the function $q(f, g, \phi)$ can be computed in time $O(k^3 2^k)$ using the following inductive formulas:

$$\begin{aligned}
&\forall f \in F: \quad q(f, f, \{f\}) = \infty \\
&\forall \phi \subset F, \phi \neq \emptyset, \forall f \in (F \setminus \phi), \forall g \in \phi: \\
&\qquad q(f, f, \phi \cup \{f\}) = \min\{s_{fh} + s(h, \phi \cup \{f\}),\ h \in \phi\} \\
&\qquad q(f, g, \phi \cup \{f\}) = \min\{s_{fh} + q(h, g, \phi),\ h \in \phi\}
\end{aligned}$$

When we do not care which family actually starts the processing we omit the first argument. Thus $q(g, \phi)$ is a minimal setup time needed for processing the activities with families ϕ under the condition that the processing of family g is interrupted at least once by an activity with type from $\phi \setminus \{g\}$. The function $q(g, \phi)$ is defined by the following formula:

$$\forall \phi \subset F, \forall g \in \phi: \quad q(g, \phi) = \min\{q(f, g, \phi),\ f \in \phi\}$$

3 Sequence Composition Rules

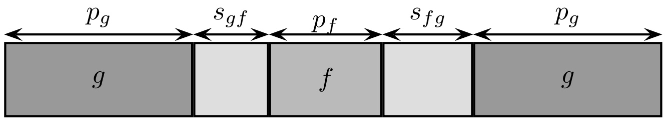
As mentioned above we try find out when the processing of one family family cannot be interrupted by an activity with different family. Consider a family g and a set of tasks Ω which contains at least two activities with the family g. If we interrupt processing of the family g by some activity $j \in \Omega$, $f_j \neq g$, then the processing of activities Ω takes at least the time:

$$v(\Omega, g) = \begin{cases} q(F_\Omega, g) + u(\Omega) & \text{when } u(\Omega, g) > p_g \\ q(F_\Omega, g) + u(\Omega) + p_g & \text{when } u(\Omega, g) = p_g \end{cases}$$

It is possible that there is not enough time for such interruption:

$$d_\Omega - r_\Omega < v(\Omega, g) \tag{1}$$

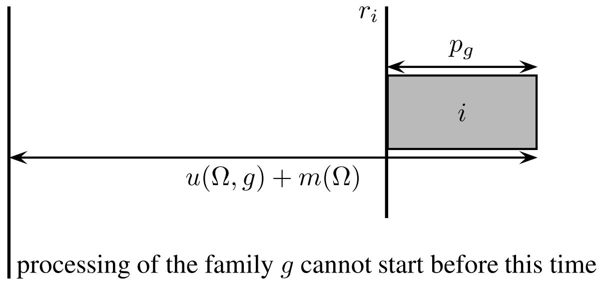
In this case all the activities from Ω with the family g have to be processed without interruption by another activities from Ω. The interruption still can occur but only by an activity from $T \setminus \Omega$ (and with the family that is not in F_Ω). The following figure shows an example of such interruption:



Anyway, processing of all the activities from Ω with the family g (including possible interruptions by activities from $T \setminus \Omega$) cannot take longer than $u(\Omega, g) + m(\Omega)$, where $m(\Omega)$ is the slack (free time) in Ω:

$$m(\Omega) = d_\Omega - r_\Omega$$

In order to process the activity i with the family g together with the other activities of the family g, we cannot start processing of the family g before $r_i + p_g - u(\Omega, g) - m(\Omega)$:



Similarly, processing of the family g cannot complete after $d_i - p_g + u(\Omega, g) + m(\Omega)$. When we combine the above deductions for all the activities of the family g from the set Ω, we get that processing of the family g can't start before the time $t_1(\Omega)$ and complete after the time $t_2(\Omega)$:

$$t_1(\Omega) = \max\{r_i,\ i \in \Omega \ \&\ f_i = g\} + p_g - u(\Omega, g) - m(\Omega)$$
$$t_2(\Omega) = \min\{d_i,\ i \in \Omega \ \&\ f_i = g\} - p_g + u(\Omega, g) + m(\Omega)$$

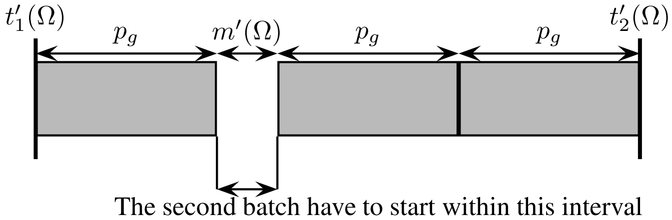
It is possible that $t_1(\Omega) < r_\Omega$ or $t_2(\Omega) > d_\Omega$. Hence we introduce the new values $t'_1(\Omega)$ and $t'_2(\Omega)$:

$$t'_1(\Omega) = \max\{t_1(\Omega), r_\Omega\}$$
$$t'_2(\Omega) = \min\{t_2(\Omega), d_\Omega\}$$

All the activities with family g in the set Ω have to be processed in the interval $\langle t'_1(\Omega), t'_2(\Omega)\rangle$. When we process only such activities in this interval then there is a slack:

$$m'(\Omega) = t'_2(\Omega) - t'_1(\Omega) - u(\Omega, g)$$

The first batch in the sequence have to start in the interval $\langle t'_1(\Omega), t'_1(\Omega) + m'(\Omega)\rangle$, the second batch in the interval $\langle t'_1(\Omega) + p_g, t'_1(\Omega) + p_g + m'(\Omega)\rangle$ etc.:



Although we do not know yet, which activity will be in the first batch etc., we can still deduce the new time windows for the activities of the family g. The new values for r_j and d_j must be in the following union of intervals:

$$r_j \in \bigcup_{l \in \mathbb{N}_0} \langle t'_1(\Omega) + lp_g, t'_1(\Omega) + lp_g + m'(\Omega) \rangle \tag{2}$$

$$d_j \in \bigcup_{l \in \mathbb{N}_0} \langle t'_2(\Omega) - lp_g - m'(\Omega), t'_2(\Omega) - lp_g \rangle \tag{3}$$

Notice that the above union of intervals is in fact a single interval modulo p_g. Let us define $\Lambda(a, b, c)$:

$$\Lambda(a, b, c) = \{a \bmod c, (a + 1) \bmod c, \ldots, b \bmod c\}$$

The set $\Lambda(a, b, c)$ is an interval in $(\mathbb{N} \bmod c)$ so we can represent it by two values only - the lower bound and the upper bound. The intersection of two such intervals $\Lambda(a, b, c)$ and $\Lambda(d, e, c)$ is again an interval in the form $\Lambda(x, y, c)$. Such intersection can be found in the time $O(1)$.

Now we can rewrite the rules (2) and (3) using Λ:

$$r_i \geq t_1(\Omega) \tag{4}$$

$$r_i \bmod p_g \in \Lambda(t'_1(\Omega), t'_1(\Omega) + m'(\Omega), p_g) \tag{5}$$

$$d_i \leq t_2(\Omega) \tag{6}$$

$$d_i \bmod p_g \in \Lambda(t'_2(\Omega) - m'(\Omega), t'_2(\Omega), p_g) \tag{7}$$

4 The Algorithm Sequence Composition

It is not necessary to apply the rules for all the sets $\Omega \subseteq T$:

Theorem 1 *To find all the changes resulting from the rules (1), (2), and (3) it is sufficient to choose all the sets Ω in the form of a tasks interval.*

Proof: Let us consider an arbitrary set Ω. Let Ψ denotes the set:

$$\Psi = \{i,\ i \in T \ \&\ r_i \geq r_\Omega \ \&\ d_i \leq d_\Omega\}$$

The set Ψ is a tasks interval and we are going to show that the above rules deduce the same or even better change of r_i for the set Ψ in comparison to set Ω. It is obvious that:

$$\Psi \supseteq \Omega$$
$$r_\Psi = r_\Omega$$
$$d_\Psi = d_\Omega$$

Hence $v(\Omega, g) \leq v(\Psi, g)$ for an arbitrary family g. Because (1) holds for Ω it have to hold for Ψ as well. From the above three formulae we deduce:

$$m(\Psi) \leq m(\Omega)$$
$$u(\Psi, g) \geq u(\Omega, g)$$
$$\max\{r_i,\ i \in \Psi \ \&\ f_i = g\} \geq \max\{r_i,\ i \in \Omega \ \&\ f_i = g\}$$
$$\min\{d_i,\ i \in \Psi \ \&\ f_i = g\} \leq \min\{d_i,\ i \in \Omega \ \&\ f_i = g\}$$
$$t_1(\Psi) \geq t_1(\Omega)$$
$$t_2(\Psi) \leq t_2(\Omega)$$
$$t'_1(\Psi) \geq t'_1(\Omega)$$
$$t'_2(\Psi) \leq t'_2(\Omega)$$
$$m'(\Psi) \leq m'(\Omega)$$

Because the difference between $u(\Psi, g)$ and $u(\Omega, g)$ cannot be bigger than the difference between $m(\Omega)$ and $m(\Psi)$, the following inequality holds: $u(\Omega, g) + m(\Omega) \geq u(\Psi, g) + m(\Psi)$.

So the changes of r_i and d_i deduced by the rules (4) and (5) applied on the set Ψ are at least as good as the changes deduced by the same rules applied on the set Ω. $\square$

When we change the value r_i using the rule (5) for the set Ω_1, it may happen than after the second change using the same rule but another set Ω_2 the new value for r_i does not match with the rule (5) and the set Ω_1. Therefore we do not change the values r_i immediately but we write down the intervals Λ_r and Λ_d for the values $(r_i \bmod p_i)$ and $(d_i \bmod p_i)$. Then we compute the intersection of all such intervals and finally we make the changes of r_i and d_i.

Let us choose an activity j and a family g. We create the sequence of all the tasks intervals $\Omega_0 \subseteq \Omega_1 \subseteq \cdots \subseteq \Omega_x$ such that $d_j = d_{\Omega_0} = d_{\Omega_1} = \cdots = d_{\Omega_x}$. Let us introduce a new notation:

$$\begin{aligned}
n(\Omega, f) &= |\{k,\ k \in \Omega \ \&\ f_k = f\}| \\
M(i) &= \{l,\ l \in \{i, i+1, \ldots, x\} \ \& \\
&\qquad d_{\Omega_l} - r_{\Omega_l} < v(\Omega_l, g) \ \&\ n(\Omega_l, g) \geq 2\} \\
t_1(i) &= \max\{t_1(\Omega_l),\ l \in M(i)\} \\
t_2(i) &= \min\{t_2(\Omega_l),\ l \in M(i)\} \\
\Lambda_r(i) &= \bigcap_{l \in M(i)} \Lambda(t_1'(\Omega_l), t_1'(\Omega_l) + m'(\Omega_l), p_g) \\
\Lambda_d(i) &= \bigcap_{l \in M(i)} \Lambda(t_2'(\Omega_l) - m'(\Omega_l), t_2'(\Omega_l), p_g)
\end{aligned}$$

For a given activity j and a family g we can compute all the values $t_1(i), t_2(i), \Lambda_r(i), \Lambda_d(i)$ in time $O(n)$.

Now consider an activity k of a family g such that $k \in \Omega_i$ and $k \notin \Omega_{i-1}$. The rules (1), (2), and (3) for the activity k and the sets Ω such as $d_\Omega = d_j$ deduce exactly:

$$\begin{aligned}
r_i &\geq t_1(i) \\
r_i \bmod p_g &\in \Lambda_r(i) \\
d_i &\leq t_2(i) \\
d_i \bmod p_g &\in \Lambda_r(i)
\end{aligned}$$

The above formulae are the basis of the filtering algorithm. We expect that all the activities are sorted in the decreasing order of r_i. The time time complexity of this algorithm is $O(kn^2)$.

```
for i ∈ T do begin
   // lr is the set of possible values r_i mod p_{f_i}:
   lr[i] := {0, 1, ..., p_{f_i} − 1} ;
   // similarly ld:
   ld[i] := {0, 1, ..., p_{f_i} − 1} ;
end;

for g ∈ F do begin
   for j ∈ T do begin
      count values t_1(i), t_2(i), Λ_r(i) and Λ_d(i) for i = 0, 1, ..., x;
      k := activity with the greatest r_i ;
      i := x;
      while i ≥ 0 and k ∈ T do begin
         if f_k ≠ g or d_k > d_j then begin
            k := next activity with the same or greater r_k ;
            continue;
         end;

         // Now we know f_k = g and k ∈ Ω_i.
         if r_k < r_{Ω_{i−1}} then begin
            // k ∉ Ω_{i−1}
            r_k ≥ t_1(i);
            d_k ≤ t_2(i);
            lr[k] := lr[k] ∩ Λ_r(i);
            ld[k] := ld[k] ∩ Λ_d(i);
            k := next activity with the same or greater r_k ;
         end else
            // We already went over all activities k such that f_k = g, k ∈ Ω_i and k ∉ Ω_{i−1}.
            // So continue with Ω_{i−1}:
            i := i−1;
      end;
   end;
end;

for i ∈ T do begin
   if lr[i] = ∅ or ld[i] = ∅ then fail ;
   r_i := the smallest integer greater or equal r_i such that r_i mod p_{f_i} ∈ lr[i];
   d_i := the greatest integer lower or equal d_i such that d_i mod p_{f_i} ∈ ld[i];
end;
```

5 Experimental Results

The individual filtering techniques have been combined into a single filtering algorithm accordingly to (Vilím and Barták 2002b). New algorithm sequence composition is in the outermost loop because most of the time it do not deduce new reductions:

repeat
  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
  sequence composition
until no more changes found

As the benchmark set we used (Vilím and Barták 2002a). For each problem we measured the computer time[1] and number of backtracks with and without the sequence composition. We measured number of reductions made by sequence composition. These reductions are of two types:

Reductions 1: from the rules (4) and (6), i.e. $r_i \geq t_1(\Omega)$ and $d_i \leq t_2(\Omega)$.
Reductions 2: from the rules (5) and (7), i.e:

$$r_i \bmod p_g \in \Lambda(t'_1(\Omega), t'_1(\Omega) + m'(\Omega), p_g)$$
$$d_i \bmod p_g \in \Lambda(t'_2(\Omega) - m'(\Omega), t'_2(\Omega), p_g)$$

Table 1 shows the results.

6 Conclusions

Experimental results are disappointing – only for problems a and p the number of backtracks is little bit lower but the CPU time bigger for all the problems. In the cases d, e, g, o, v and w the algorithm made some reductions but the number of backtracks stayed the same – sequence composition found some reductions, but without the sequence composition the same reductions were deduced also by another filtering algorithm somewhere deeper in the search tree.

Reduction of type 1 was not found for any of the problems. It indicates that the benchmark set is not really random. The question is how to generate better benchmark problems.

References

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

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*, 2002b.

[1] On Intel Pentium Celeron 375MHz

						with sequence composition			
problem	n	k	solutions	backtracks	time	backtracks	time	Reductions 1.	Reductions 2.
a	30	3	88	12	2.22s	9	2.60s	0	2
b	25	5	56251	5016	25m 30s	5016	30m 52s	0	0
c	25	5	72	0	1.78s	0	2.11s	0	0
d	40	6	12	1	1.02s	1	1.15s	0	1
e	20	2	28	0	0.20s	0	0.26s	0	1
f	50	6	6	2	0.70s	2	0.86s	0	0
g	30	3	1690	77	37.08s	77	45.23	0	5
h	75	5	12	2	2.90s	2	3.40s	0	0
i	50	5	48	44	7.59s	44	8.89s	0	0
j	50	5	10	4	1.34s	4	1.43s	0	0
k	50	7	9	0	1.33s	0	1.46s	0	0
l	50	5	4	0	0.51s	0	0.52s	0	0
m	50	3	3	3	0.35s	3	0.40s	0	0
n	30	5	39	13	1.78s	13	1.94s	0	0
o	50	5	32	8	3.60s	8	4.14s	0	0
p	30	3	270	24	6.05s	18	6.69s	0	13
q	50	7	228	0	28.03s	0	33.90s	0	0
r	50	7	324	0	44.94s	0	49.01s	0	0
s	100	7	50	0	26.48s	0	30.74s	0	0
t	200	7	8	0	18.00s	0	20.36s	0	0
v	100	7	240	0	2m 6s	0	2m 25s	0	1
w	50	2	24	4	1.36s	4	1.50s	0	1
x	50	2	1368	384	1m 8s	384	1m 18s	0	0
z	50	4	24	7	2.14s	7	2.58s	0	0

Table 1: Experimental results.