SOLVING LARGE CUMULATIVE SCHEDULING PROBLEMS

ARNAUD LETORT,¹ NICOLAS BELDICEANU,¹ AND MATS CARLSSON²

arnaud.letort@mines-nantes.fr, nicolas.beldiceanu@mines-nantes.fr, matsc@sics.se

¹EMN, TASC (CNRS/INRIA)
²SICS

APRIL 16, 2013 (PGMO SEMINAR)
MOTIVATIONS

• Need to handle large scale problems.
  [Panel of the Future of CP 2011]

• (Multi-dimensional) bin-packing problems, in the context of cloud computing.
  [Panel of the Future of CP 2011], [2012 Roadef Challenge]

• Existing papers usually leave open the scalability issue.

• Time-Table constraint is a good candidate.
  [Baptiste 2006, Samos] time-tabling used in ILOG Scheduler for scalability purpose
  [Vilim, 2011 CPAIOR]
OUTLINE

The *cumulative* Constraint

A Critical Analysis of the [CP2001] Sweep Algorithm

The Dynamic Sweep Algorithm (one *cumulative*)

The Dynamic Sweep Algorithm (several *cumulative + precedence*)

Evaluation

Conclusion
THE CUMULATIVE CONSTRAINT

4 Tasks
OUTLINE

The *cumulative* Constraint

A Critical Analysis of the [CP2001] Sweep Algorithm
  • Principle
  • Illustration
  • 4 Weaknesses

The Dynamic Sweep Algorithm (one *cumulative*)

The Dynamic Sweep Algorithm (several *cumulative* + *precedence*)

Evaluation

Conclusion
A CRITICAL ANALYSIS OF THE [CP2001] SWEEP ALGORITHM
(OVERVIEW)

The sweep-line “jumps” from event to event in order to build the cumulated profile and to perform checks and pruning.
A CRITICAL ANALYSIS OF THE [CP2001] SWEEP ALGORITHM
(PRINCIPLE: COMPULSORY PARTS)

**Compulsory Part**: the intersection of all the feasible instances of a task.

**Cumulated Profile**: the union of all the compulsory parts.
A CRITICAL ANALYSIS OF THE [CP2001] SWEEP ALGORITHM
(PRINCIPLE: EVENTS)

Event: a potential change of the height of the cumulated profile. (i.e. start and end of compulsory part)
1. All events on the current sweep-line position are read (amount of available resource is updated).

2. The current and the next sweep-line positions define a sweep interval.

3. Scans all tasks that overlap the sweep interval. If the height of a task is greater than the available resource, an interval is removed from the start of the task.
A CRITICAL ANALYSIS OF THE [CP2001] SWEEP ALGORITHM
(ILLUSTRATION: INITIAL PROBLEM)

\[ t_0 : s_0 = 0, \ d_0 = 1, \ e_0 = 1, \ h_0 = 3 \]
\[ t_1 : s_1 \in [0, 2], \ d_1 = 2, \ e_1 \in [2, 4], \ h_1 = 3 \]
\[ t_2 : s_2 \in [2, 4], \ d_2 = 3, \ e_2 \in [5, 7], \ h_2 = 3 \]
\[ t_3 : s_3 \in [5, 7], \ d_3 = 1, \ e_3 \in [6, 8], \ h_3 = 3 \]
A CRITICAL ANALYSIS OF THE [CP2001] SWEEP ALGORITHM
(ILLUSTRATION: AFTER A FOURTH SWEEP)

After 4 sweeps

\[ t_0 : s_0 = 0, \, d_0 = 1, \, e_0 = 1, \, h_0 = 3 \]
\[ t_1 : s_1 \in [1, 2], \, d_1 = 2, \, e_1 \in [3, 4], \, h_1 = 3 \]
\[ t_2 : s_2 \in [3, 4], \, d_2 = 3, \, e_2 \in [6, 7], \, h_2 = 3 \]
\[ t_3 : s_3 \in [6, 7], \, d_3 = 1, \, e_3 \in [7, 8], \, h_3 = 3 \]
A CRITICAL ANALYSIS OF THE [CP2001] SWEEP ALGORITHM
(4 WEAKNESSES)

1. **Too static:**
   
   Does not take into account the potential increase of the cumulated profile during a single sweep (see previous example).

2. **Often reaches its worst time complexity:**
   
   It needs to systematically re-scan all tasks that overlap the current sweep-line position to perform pruning. (O(n²))

3. **Creates holes in the domains:**
   
   A variable cannot just be compactly represented by its min/max values.

4. **Does not take advantage of the bin-packing:**
   
   The worst-case time complexity is left unchanged and is often reached.
OUTLINE

The cumulative Constraint

A Critical Analysis of the [CP2001] Sweep Algorithm

The Dynamic Sweep Algorithm (one cumulative)
  • Principle
  • Illustration
  • Property and Complexity
  • Greedy Mode

The Dynamic Sweep Algorithm (several cumulative + precedence)

Evaluation

Conclusion
THE DYNAMIC SWEEP ALGORITHM (PRINCIPLE)

1. A Dynamic sweep based algorithm:
   It can directly take into account the increase of the cumulated profile during a single sweep.

2. A “good” average time complexity:
   Essential in order to handle large instances.

3. Does not create holes in domains:
   A variable can be compactly represented by its min/max values.

4. Takes advantage of the bin-packing:
   A better worst-case time complexity than for the cumulative.
THE DYNAMIC SWEEP ALGORITHM
(PRINCIPLE)

• Deal with domain bounds. [CP2012]
  (Creates holes in the domains. [CP2001])

• Filter min and max values in two distinct sweep stages:
  sweep_min and sweep_max, speeds up the convergence
to the fixpoint. [CP2012]

• New dynamic and conditional events [CP2012]
  (Too static [CP2001])

• Use dedicated data structures. [CP2012]
  (Often reaches its worst time complexity [CP2001])
THE DYNAMIC SWEEP ALGORITHM
(PRINCIPLE: NEW EVENTS)

• Event related to the end of the compulsory part of a task is now dynamic.

• A conditional event is generated for each task initially without compulsory part.

  The adjustment of the earliest start of the task can induce the creation of a compulsory part.

  The conditional event is transformed into 2 events reflecting the new compulsory part.
To partially avoid rescanning of all tasks:

- A heap $h_{\text{conflict}}$ storing tasks in conflict with the current sweep interval. Tasks are ordered by increasing height.

- A heap $h_{\text{check}}$ storing tasks not in conflict on the current sweep interval and for which the earliest start is not yet found. Tasks are ordered by decreasing height.
sweep interval = [0,1)
available resource = 2

\( h_1 (=3) \) is greater than the available resource (=2).

\( t_1 \) is added into \( h_{\text{conflict}} \).
THE DYNAMIC SWEEP ALGORITHM
(ILLUSTRATION)

sweep interval = [1,2)
available resource = 5

Top task of $h_{\text{conflict}} (t_1)$ is not greater than the available resource. Consequently $t_1$ is removed from $h_{\text{conflict}}$ and added into $h_{\text{check}}$.

Earliest start of $t_1$ is adjusted to 1. Its conditional event is transformed into 2 events reflecting its new compulsory part.
sweep interval = [2,3)
available resource = 2

$h_2 (=3)$ is greater than the available resource (=2).

t_2 is added into $h_{\text{conflict}}$. 
THE DYNAMIC SWEEP ALGORITHM
(ILLUSTRATION)

sweep interval = [3,4)
available resource = 5

Top task of $h_{\text{conflict}} (t_2)$ is not greater than the available resource. Consequently $t_2$ is removed from $h_{\text{conflict}}$ and added into $h_{\text{check}}$.

Earliest start of $t_2$ is adjusted to 3. Event related to its end of compulsory part is pushed from 5 to 6.
THE DYNAMIC SWEEP ALGORITHM
(ILLUSTRATION)

sweep interval = [4,5)
available resource = 2

Nothing to do.
THE DYNAMIC SWEEP ALGORITHM (ILLUSTRATION)

sweep interval = [5,6)
available resource = 2

$h_3 (=3)$ is greater than the available resource (=2).

t_3 is added into $h_{conflict}$. 
sweep interval = [6, 7)
available resource = 5

Top task of $h_{conflict}(t_3)$ is not greater than the available resource. Consequently $t_3$ is removed from $h_{conflict}$.

Earliest start of $t_3$ is adjusted to 6. Nothing else to do.
THE DYNAMIC SWEEP ALGORITHM
(PROPERTY AND COMPLEXITY)

• A worst-case time complexity of $O(n^2 \log n)$ where $n$ is the number of tasks.
  There is a variant with a worst-case time complexity of $O(n^2)$,
  but the $O(n^2 \log n)$ version scales better.

• Property after a call to sweep_min:

For any task $t$ in $T$, one can schedule $t$ at its earliest start without exceeding the resource limit wrt. the cumulated profile of $T\backslash\{t\}$. 
THE DYNAMIC SWEEP ALGORITHM
(GREEDY MODE USING FILTERING)

Why?
To handle larger (10 million tasks) instances in a CP solver.

How?
It reuses the sweep_min part but directly fixes the start of the task rather than adjusting it. Then, the sweep-line is reset to this start and the process continues until all tasks get fixed or a resource overflow occurs.
The *cumulative* Constraint

A Critical Analysis of the [CP2001] Sweep Algorithm

The Dynamic Sweep Algorithm (one *cumulative*)

The Dynamic Sweep Algorithm (several *cumulative* + *precedence*)

Evaluation

Conclusion
KEY IDEAS

Handle all cumulative and precedence in one pass (adjusting min)

Reuse the idea used just for a set of precedence (topological ordering) but also consider the cumulative constraints while propagating

Each constraint is not propagated in isolation (i.e. independently from the other) but gradually

1. Sort the variables wrt their min,
2. Propagate all constraints just on the first variable (unless its min increases and it is not the first variable anymore)
3. Discard first variable and continue …
EVENTS TYPES

(SCP,t,latest start): Start of Compulsory part (even if no CP)

(ECPD,t,earliest end): End of Compulsory part (only if CP)

(PR,t,earliest start): Pruning (only if not fixed)

(RS,t,earliest end): Release Successors (at least one successor)

prevent the earliest starts of the successor of t
from being adjusted before the final earliest start
of task t has been determined
tasks on resource $r_1$

$\begin{align*}
 t_0 & : [1, 1] \\
 t_1 & : [0, 3] \\
 t_2 & : [0, 5] \\
 t_3 & : [0, 9]
\end{align*}$

precendece graph

$\begin{align*}
 t_2 & \rightarrow t_4 \\
 t_1 & \rightarrow 1 \\
 t_0 & \rightarrow t_3 \\
 t_0 & \rightarrow 2
\end{align*}$

tasks on resource $r_0$

$\begin{align*}
 t_0 & : [1, 1] \\
 t_1 & : [0, 3] \\
 t_2 & : [0, 5] \\
 t_3 & : [0, 9] \\
 t_4 & : [0, 7]
\end{align*}$

resource $r_1$

$\Delta (gap = 2)$

$\leq 2$

time

resource $r_0$

$\Delta (gap = 3)$

$\leq 3$

time

EVENTS (before)

$\begin{align*}
e_1 & : \langle 0, t_1, PR \rangle \\
e_2 & : \langle 0, t_2, PR \rangle \\
e_3 & : \langle 1, t_0, SCP \rangle \\
e_4 & : \langle 2, t_0, ECPD \rangle \\
e_5 & : \langle 2, t_0, RS \rangle \\
e_6 & : \langle 2, t_1, RS \rangle \\
e_7 & : \langle 2, t_2, RS \rangle \\
e_8 & : \langle 3, t_1, SCP \rangle \\
e_9 & : \langle 5, t_2, SCP \rangle
\end{align*}$

STATUS (before)

$\begin{align*}
t_0 & : \text{ready} \\
t_1 & : \text{none} \\
t_2 & : \text{none} \\
t_3 & : \text{none} \\
t_4 & : \text{none}
\end{align*}$

Fig. 1: initialization and processing events $\langle 0, t_1, PR \rangle$ and $\langle 0, t_2, PR \rangle$
Fig. 2: after initialization and processing events \( \langle 0, t_1, \text{PR} \rangle \) and \( \langle 0, t_2, \text{PR} \rangle \)
Fig. 3: filter min wrt $[\Delta = 0, \Delta_{next} = 1]$: no pruning
Fig. 4: process event \(\langle 1, t_0, \text{SCP} \rangle\)
Fig. 5: filter min wrt \([\Delta = 1, \Delta_{next} = 2]\) after processing event \((1, t_0, SCP)\)
Fig. 6: process event \( \langle 2, t_0, \text{ECPD} \rangle \) (gaps increase since end of compulsory part)
tasks on resource $r_1$

- $t_0$: [1, 1]
- $t_1$: [0, 3]
- $t_2$: [0, 5]
- $t_3$: [0, 9]

tasks on resource $r_0$

- $t_0$: [1, 1]
- $t_1$: [0, 3]
- $t_2$: [0, 5]
- $t_3$: [0, 9]
- $t_4$: [0, 7]

precedence graph

$\rightarrow$

resource $r_1$

\[\Delta \text{ (gap = 2)}\]

resource $r_0$

\[\Delta \text{ (gap = 3)}\]

EVENTS

- $e_4$: \(\langle 2, t_0, \text{ECPD} \rangle\)
- $e_5$: \(\langle 2, t_0, \text{RS} \rangle\)
- $e_6$: \(\langle 2, t_1, \text{RS} \rangle\)
- $e_7$: \(\langle 2, t_2, \text{RS} \rangle\)
- $e_8$: \(\langle 3, t_1, \text{SCP} \rangle\)
- $e_9$: \(\langle 5, t_2, \text{SCP} \rangle\)

STATUS

- $t_0$: ready
- $t_1$: conflict($r_0$)
- $t_2$: conflict($r_1$)
- $t_3$: none
- $t_4$: none

Fig. 7: process event \(\langle 2, t_0, \text{RS} \rangle\) ($t_0$ has reached its final earliest end)
tasks on resource \( r_1 \)

- \( t_0 \): [1, 1]  
- \( t_1 \): [0, 3]  
- \( t_2 \): [0, 5]  
- \( t_3 \): [0, 9]  

tasks on resource \( r_0 \)

- \( t_0 \): [1, 1]  
- \( t_1 \): [0, 3]  
- \( t_2 \): [0, 5]  
- \( t_3 \): [0, 9]  
- \( t_4 \): [0, 7]  

precedence graph

- \( t_2 \) → \( t_4 \)
- \( t_1 \) → \( t_3 \)

resource \( r_1 \)

- \( t_0 \)
- \( \Delta \text{ (gap = 2)} \)

resource \( r_0 \)

- \( t_0 \)
- \( \Delta \text{ (gap = 3)} \)

EVENTS

- \( e_4: (2, t_0, \text{ECPD}) \)
- \( e_5: (2, t_0, \text{RS}) \)
- \( e_6: (2, t_1, \text{RS}) \)
- \( e_7: (2, t_2, \text{RS}) \)
- \( e_8: (3, t_1, \text{SCP}) \)
- \( e_9: (5, t_2, \text{SCP}) \)

STATUS

- \( t_0 \): ready
- \( t_1 \): conflict(\( r_0 \))
- \( t_2 \): conflict(\( r_1 \))
- \( t_3 \): none
- \( t_4 \): none

Fig. 8: process events \( \langle 2, t_1, \text{RS} \rangle, \langle 2, t_2, \text{RS} \rangle \) (\( t_1, t_2 \) do not have yet reached their final earliest end, so both events are shifted)
Fig. 9: filter min wrt \(\Delta = 2, \Delta_{next} = 3\) after processing events at date 2 (create compulsory part for \(t_1\) and corresponding event for end of compulsory part)
Fig. 10: process event \( \langle 3, t_1, SCP \rangle \) (find final earliest start for \( t_1 \))
Fig. 11: filter min wrt $[\Delta = 3, \Delta_{next} = 4]$ after processing events at date 3 ($\langle 3, t_1, SCP \rangle$)
Fig. 12: process event $\langle 4, t_1, \text{ECPD} \rangle$ (gaps increase since end of compulsory part)
Fig. 13: process event \( \langle 4, t_1, \text{RS} \rangle \) (\( t_1 \) has reached its final earliest end, its successor
**Fig. 14:** process event \(4, t_2, \text{RS}\) \(t_2\) has not yet reached its final earliest end, consequently shift the RS event.
Fig. 15: process event \(\langle 4, t_3, \text{PR}\rangle\) \((t_3\) can be include in \([\Delta = 4, \Delta_{\text{next}} = 5]\) so \(t_3\) has reached its final earliest start)
tasks on resource $r_1$

$\begin{array}{ll}
    t_0 & [1, 1] \\
    t_1 & [2, 3] \\
    t_2 & [4, 5] \\
    t_3 & [4, 9] \\
\end{array}$

resource $r_1$

$\Delta (gap = 2)$

tasks on resource $r_0$

$\begin{array}{ll}
    t_0 & [1, 1] \\
    t_1 & [2, 3] \\
    t_2 & [4, 5] \\
    t_3 & [4, 9] \\
    t_4 & [9, 7] \\
\end{array}$

resource $r_0$

$\Delta (gap = 3)$

precedence graph

$\begin{array}{ll}
    t_2 & 0 \\
    t_1 & 0 \\
    t_0 & 0 \\
    t_3 & 0 \\
    t_4 & 1 \\
\end{array}$

EVENTS

$\begin{array}{ll}
    e_{11} & \langle 4, t_3, PR \rangle \\
    e_9 & \langle 5, t_2, SCP \rangle \\
    e_{13} & \langle 6, t_2, ECPD \rangle \\
    e_7 & \langle 6, t_2, RS \rangle \\
    e_{12} & \langle 9, t_3, SCP \rangle \\
\end{array}$

STATUS

$\begin{array}{ll}
    t_0 & \text{ready} \\
    t_1 & \text{ready} \\
    t_2 & \text{conflict}(r_1) \\
    t_3 & \text{ready} \\
    t_4 & \text{none} \\
\end{array}$

(date, task, type)

Fig. 16: filter min wrt $[\Delta = 4, \Delta_{next} = 5]$ after processing events at date 4 (create compulsory part for $t_2$ and corresponding event for end of compulsory part)
OUTLINE

The *cumulative* Constraint

A Critical Analysis of the [CP2001] Sweep Algorithm

The Dynamic Sweep Algorithm (one *cumulative*)

The Dynamic Sweep Algorithm (several *cumulative + precedence*)

Evaluation

Conclusion
EVALUATION

- Random instances with a density close to 0.7.
- A speedup increasing with the number of tasks.
- The dynamic is more robust than the 2001 sweep wrt. different heuristics.
- The greedy mode could handle:
  - 1 million tasks in 12 minutes
  - up to $10^7$ tasks in ~8h (swap).
OUTLINE

The cumulative Constraint

A Critical Analysis of the [CP2001] Sweep Algorithm

The Dynamic Sweep Algorithm (one cumulative)

The Dynamic Sweep Algorithm (several cumulative + precedence)

Evaluation

Conclusion
CONCLUSION

• a **lean** sweep based filtering algorithm

• **dynamically** handle creation/extension of CP

• **faster** and **more scalable** than the 2001 sweep

• handle up to **10 million tasks** in greedy mode.
  handle up to **1 million tasks** with **2 millions precedences** and **64 resources** in greedy mode.
MORE INFORMATION

CP 2012 paper:
One single cumulative constraint,
LNCS vol. 7514, pp. 439-454

CPAIOR 2013 paper:
several cumulative constraints
Available at http://www.cis.cornell.edu/ics/cpaior2013/papers.php

Technical report (April 2013)
Several cumulative + precedence constraints
Available at http://soda.swedish-ict.se/5504/