

New Filtering for the *cumulative* Constraint in the Context of Non-Overlapping Rectangles

N. Beldiceanu, LINA UMR CNRS 6241; M. Carlsson, SICS;
E. Poder, LINA UMR CNRS 6241



May 16, 2008

Outline

Overview

The Longest Cumulative Hole Problem

Balancing Knapsack Constraints

Performance Evaluation

Conclusion

Context

- ▶ 2 dimensional non-overlapping (*but most results are also reused for $k > 2$ dimensions in geost*)
- ▶ Focus on exact methods (*proving that no solution exist*)
- ▶ Challenge: can we find *all* solutions for some perfect squares placement problems just by propagating?

Why *cumulative* ?

(and not just *non-overlapping*)

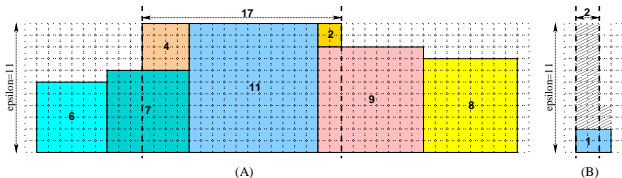
- ▶ Well-known necessary condition even when $k > 2$ (*cumulative* in dimension d):
 - ▶ **origin** of a task: *coord. in dimension d*
 - ▶ **duration** of a task: *size in dimension d*
 - ▶ **resource consumption**: *product of the sizes in dimensions different from d*
 - ▶ **overall capacity**: *product of available space in dimensions different from d*
- ▶ Well-known heuristic which labels on all coordinates in one dimension
 - ▶ (see paper by Simonis and O'Sullivan at the BPPC workshop this afternoon)

What is it all about ?

- ▶ Two complementary methods for anticipating the creation of holes
- ▶ Two kind of holes are considered:
 - ▶ Holes *on top* of an interval of the compulsory part resource profile.
 - ▶ Holes at *two neighboring time-points*.

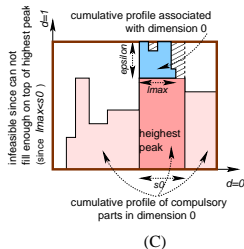
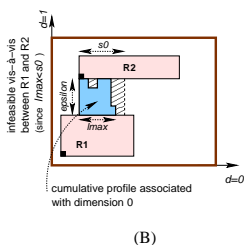
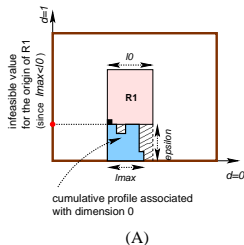
Definition of the *cumulative longest hole* Problem

- ▶ Given a *cumulative*(\mathcal{T}, L) constraint, let σ denote the difference between the *available space* and the *needed space*.
- ▶ Given an integer $\epsilon \in [1, L]$ and $\mathcal{T}' \subseteq \mathcal{T}$ for which the resource consumption is at most ϵ , the *longest hole problem* is to find the largest integer $lmax_{\sigma}^{\epsilon}(\mathcal{T}')$ such that: *there is a cumulative placement of maximum height ϵ involving a subset of tasks of \mathcal{T}' where, on one interval $[i, i + lmax_{\sigma}^{\epsilon}(\mathcal{T}') - 1]$, the area of the empty space does not exceed σ .*



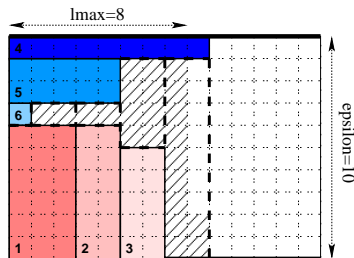
Usage of the *cumulative longest hole* in the context of *non-overlapping*

- ▶ *Vis-à-vis* between a rectangle and one of the borders,
- ▶ *Vis-à-vis* between two rectangles,
- ▶ *Highest peak* of the cumulative profile of compulsory part.



Polynomial Case

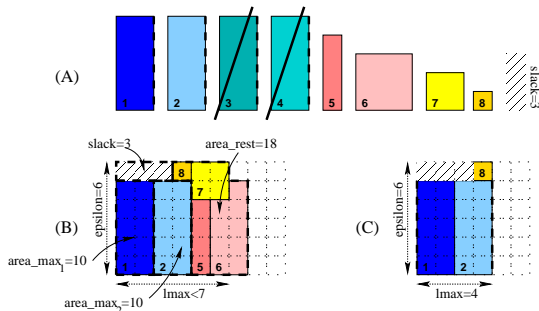
Characterisation: sum of the heights of the non-disjunctive tasks + the height of the highest disjunctive task \leq resource limit.



A First Upper Bound

Based on the following observation:

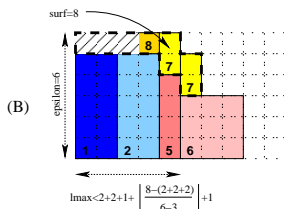
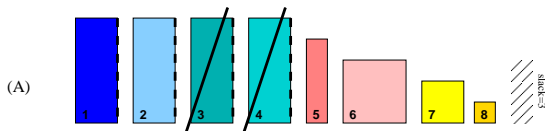
- You can cut at most two disjunctive tasks (*one for each border of the longest cumulative hole*); consequently, you should only keep the two biggest contributions in area of the disjunctive tasks that will have to be cut.



A Second Upper Bound

Based on the following observation:

- If you have *many* disjunctive tasks you may not use *all of them* if you don't have enough small tasks (and slack) to fill the space on top of these disjunctive tasks.



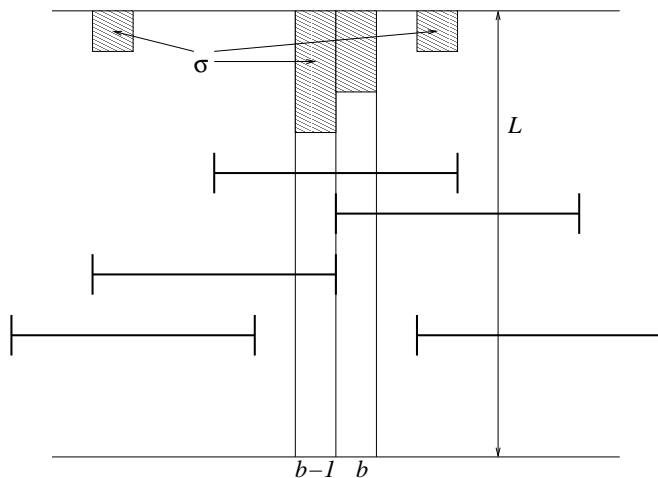
Remarks on the Bounds

- ▶ The two bounds are incomparable.
- ▶ We have other bounds (*but they rarely produce significant improvement*).

All these bounds are computed when we **post** geost for all possible gap sizes (*no overhead during the search, but we lose accuracy if the heuristic places some small objects first*).

Intuition

What tasks can overlap $b - 1, b$ with slack σ and limit L ?



Necessary Conditions

Given constraint $C \equiv \text{cumulative}(\mathcal{T}, L)$, $n = |\mathcal{T}|$, and slack σ .
 For every pair $(b-1, b)$ inside the timespan of C :

$$H_i \stackrel{\text{def}}{=} \sum \{t.h \mid t \in \mathcal{T} \wedge t \text{ intersects } i\}$$

$$H_{b-1} \in [L - \sigma, L]$$

$$H_b \in [L - \sigma, L]$$

$$H_{b-1} + H_b \in [2L - \sigma, 2L]$$

0-1 Variables

Four Mutually Exclusive Cases

- ▶ Let $t_i, i \in [1, n]$ denote the i^{th} task of \mathcal{T} .
- ▶ Introduce 0-1 variables $S_{ib}, C_{ib}, O_{ib}, N_{ib}$.
- ▶ $S_{ib} + C_{ib} + O_{ib} + N_{ib} = 1$.

| | | |
|--------------|--|------------------------------------|
| $S_{ib} = 1$ | t_i intersects b but not $b - 1$ | $t_i.s = b$ |
| $C_{ib} = 1$ | t_i intersects $b - 1$ but not b | $t_i.s = b - t_i.d$ |
| $O_{ib} = 1$ | t_i intersects both $b - 1$ and b | $t_i.s \in [b - t_i.d + 1, b - 1]$ |
| $N_{ib} = 1$ | t_i intersects neither $b - 1$ nor b | $t_i.s \notin [b - t_i.d, b]$ |

(1)

Pseudo-Boolean Equation System

$$\begin{aligned}
 \forall i \in [1, n] : S_{ib} + C_{ib} + O_{ib} + N_{ib} &= 1 \\
 H_{b-1} &= \sum_{i \in [1, n]} t_i \cdot h \cdot (C_{ib} + O_{ib}) \in [L - \sigma, L] \\
 H_b &= \sum_{i \in [1, n]} t_i \cdot h \cdot (S_{ib} + O_{ib}) \in [L - \sigma, L] \\
 H_{b-1} + H_b &\in [2L - \sigma, 2L]
 \end{aligned}
 \tag{2}$$

Dynamic Programming Recursion

$$f(k, l, r) \in \{0, 1\}$$



M.A. Trick.

A dynamic programming approach for consistency and propagation for knapsack constraints.

Annals of Operations Research, 118(1–4):73-84, 2003.

$$f(k, l, r) = \begin{cases} 1, & \text{if } \left(\begin{array}{l} \forall i \in [1, k] : S_{ib} + C_{ib} + O_{ib} + N_{ib} = 1 \wedge \\ \sum_{i \in [1, k]} t_{j \cdot h} \cdot (C_{ib} + O_{ib}) = l \wedge \\ \sum_{i \in [1, k]} t_{j \cdot h} \cdot (S_{ib} + O_{ib}) = r \end{array} \right) \\ 0, & \text{otherwise} \end{cases}$$

Dynamic Programming Recursion

Base and Recursive Cases

$$f(0, l, r) = \begin{cases} 1, & \text{if } l = 0 \wedge r = 0 \\ 0, & \text{otherwise} \end{cases}$$

$$f(k, l, r) = \max \begin{cases} f(k-1, l, r - t_k.h) \\ f(k-1, l - t_k.h, r) \\ f(k-1, l - t_k.h, r - t_k.h) \\ f(k-1, l, r) \end{cases}, \text{ if } k > 0 \quad (3)$$

(2) has a solution if and only if:

$$\exists l, r : \begin{cases} f(n, l, r) = 1 \wedge \\ l \in [L - \sigma, L] \wedge \\ r \in [L - \sigma, L] \wedge \\ l + r \in [2L - \sigma, 2L] \end{cases}$$

Dynamic Programming Recursion

Digraph View

- ▶ Nodes: (k, l, r) corresponding to $f(k, l, r) = 1$.
- ▶ Arcs: correspond to (3), annotated with the 0-1 variable that takes the value 1.

$$(k-1, l, r - t_k \cdot h) \xrightarrow{S_{kb}=1} (k, l, r)$$

$$(k-1, l - t_k \cdot h, r) \xrightarrow{C_{kb}=1} (k, l, r)$$

$$(k-1, l - t_k \cdot h, r - t_k \cdot h) \xrightarrow{O_{kb}=1} (k, l, r)$$

$$(k-1, l, r) \xrightarrow{N_{kb}=1} (k, l, r)$$

Filtering

Solutions

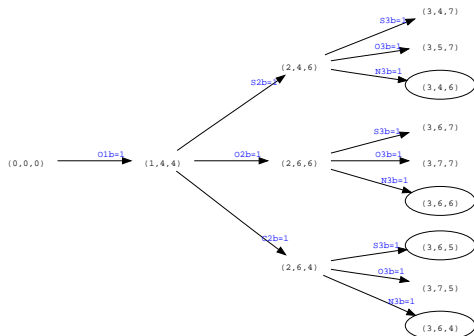
- ▶ Source node: $(0, 0, 0)$.
- ▶ Sink node: (n, l, r) where $l \in [L - \sigma, L] \wedge r \in [L - \sigma, L] \wedge l + r \in [2L - \sigma, 2L]$.
- ▶ Solution to (2): a path from source to sink.

Filtering

- ▶ Inspect all solutions.
- ▶ For each $S_{ib}, C_{ib}, O_{ib}, N_{ib}$ that is 0 in all solutions, prune $t_i.s$ according to (1).

Example with $b = 4, L = 6, \sigma = 4$

$$\begin{aligned} \mathcal{D}(t_1.s) &= \{3\}, & t_1.h &= 4, & t_1.d &= 3 \\ \mathcal{D}(t_2.s) &= \{2, 3, 4\}, & t_2.h &= 2, & t_2.d &= 2 \\ \mathcal{D}(t_3.s) &= \{1, 3, 4\}, & t_3.h &= 1, & t_3.d &= 2 \end{aligned}$$



- ▶ $O_{3b} = 0$ in all solutions.
- ▶ t_3 cannot intersect both 3 and 4, hence $t_{j.s} \neq 3$.

Strengthening the Method

Adding More Knapsack Constraints

- ▶ Identify some set $\mathcal{T}' \subseteq \mathcal{T}$ such that if $O_{ib} = 0$ for all $t_i \in \mathcal{T}'$, L is exceeded somewhere.
- ▶ Add the knapsack constraint $\sum_{t_i \in \mathcal{T}'} O_{ib} \geq 1$.

Learning Solutions

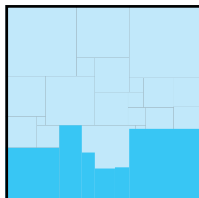
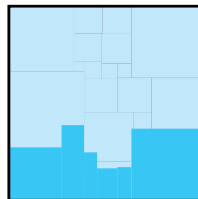
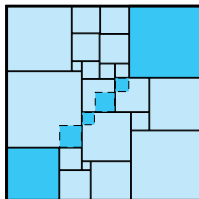
- ▶ Let the *pre-signature* be the set of 0-1 variables for which the value 1 is feasible according to (1) prior to solving.
- ▶ Let the *post-signature* be the set of 0-1 variables for which the value 1 is still feasible after solving.
- ▶ The pre-signature abstracts away from the given b as well as from variable domains.
- ▶ Caching pre/post-signature pairs saves 75% of run time.

Typical Propagation Pattern: best case

(down the leftmost path of the search tree - prob.021)

STEP 1: All compulsory parts along the main diagonal are inferred (symmetry of first square).

STEP 2: The full solution is inferred as soon as we label the second variable.



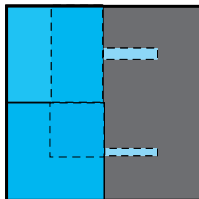
Typical Propagation Pattern: worst case

(down the leftmost path of the search tree - prob.048)

STEP 1: Identify only part of the c.p. of the biggest square.

STEP 2: Fix the biggest square and identify a part of the c.p. of the second biggest square.

STEP 3-7: Label on several x-coordinates and discover a failure.

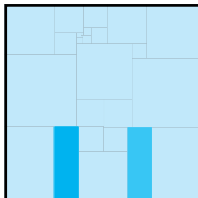
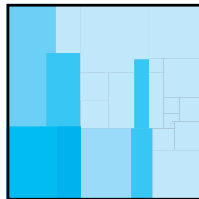
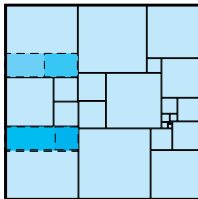


Typical Propagation Pattern: intermediate case

(down the leftmost path of the search tree - prob.076)

STEPS 1-5: Identify parts of the compulsory part of the first two squares.

STEP 6: Extend to the full solution.



Experimental Setup

Software & Hardware

- ▶ 3GHz Pentium IV, 1MB cache.
- ▶ Benchmark: Perfect Squares of order 21-25.



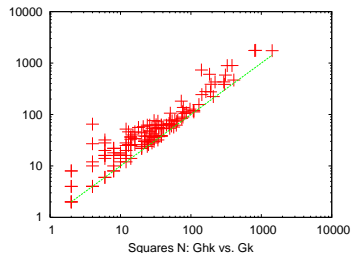
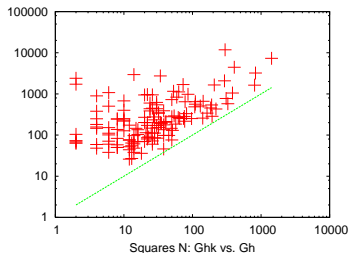
Search Procedure

1. For each square o , narrow by binary search the domain of $o.x$ until it has a compulsory part that is at least half the length of o .
2. For each square o , fix $o.x$ by binary search.
3. Repeat steps 1-2 for the Y coordinates.

Perfect Squares, All Solutions

Backtracks, Zero Slack Case

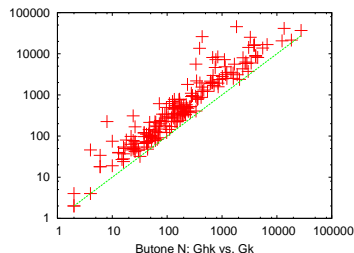
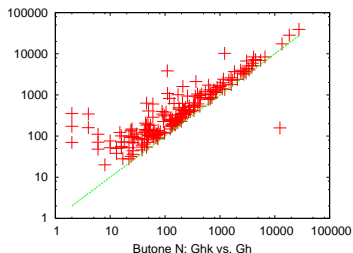
- ▶ Left: knocking out Balancing Knapsack Constraints.
- ▶ Right: knocking out Longest Cumulative Hole.



Perfect Squares, All Solutions

Backtracks, Nonzero Slack Case: All But Smallest Square

- ▶ Left: knocking out Balancing Knapsack Constraints.
- ▶ Right: knocking out Longest Cumulative Hole.



Conclusion

(*current situation*)

- ▶ Answer to challenge: yes (*surprise*).
- ▶ Context of these results:
 - ▶ No or very small slack.
 - ▶ Not too many small identical objects.
- ▶ Observation: in the context of geometrical constraints, methods that create holes are an important communication channel for knapsack constraints (*contradicting the conventional wisdom that interval reasoning is enough*).

Conclusion

(future)

- ▶ We need for more powerful methods when the slack is not so small, because:
 - ▶ It kills the knapsack constraint (*since all the slack can be located on one single column*).
 - ▶ It weakens the longest cumulative hole (*since all the slack can be located on one single interval*).
- ▶ We may consider exact methods for solving the longest hole problem for not too big gaps (*before starting the backtrack search*).