Annals of Operations Research
Manuscript Draft

Manuscript Number: ANOR-660

Title: New Filtering for the Cumulative Constraint in the Context of Non-Overlapping Rectangles

Article Type: S.I.: Integration of AI/Constraint Prog.

Keywords: Geometric Constraint; Bin-Packing; Global Constraint; Constraint Programming; Sweep

Abstract: This article describes new filtering methods for the cumulative constraint.
The first method introduces the so called longest closed hole and longest open hole problems.
For these two problems it first provides bounds and exact methods and then shows how
to use them in the context of the non-overlapping constraint.
The second method introduces balancing knapsack
constraints which relate the total height of the tasks that end at a
specific time-point with the total height of the tasks that start at
the same time-point. Experiments on tight rectangle packing problems
show that these methods drastically reduce both the time and the
number of backtracks for finding all solutions as well as for finding
the first solution. For example, we found without backtracking all
solutions to 65 perfect square instances of order 22-25 and sizes
ranging from 192x192 to 661x661.

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

Nicolas Beldiceanu[1], Mats Carlsson[2], Sophie Demassey[1], and Emmanuel Poder[1]

[1] École des Mines de Nantes, LINA UMR CNRS 6241, FR-44307 Nantes, France
{Nicolas.Beldiceanu,Sophie.Demassey,Emmanuel.Poder}@emn.fr
[2] SICS, P.O. Box 1263, SE-164 29 Kista, Sweden
Mats.Carlsson@sics.se

Abstract. This article describes new filtering methods for the *cumulative* constraint. The first method introduces the so called *longest closed hole* and *longest open hole* problems. For these two problems it first provides bounds and exact methods and then shows how to use them in the context of the *non-overlapping* constraint. The second method introduces *balancing knapsack constraints* which relate the total height of the tasks that end at a specific time-point with the total height of the tasks that start at the same time-point. Experiments on tight rectangle packing problems show that these methods drastically reduce both the time and the number of backtracks for finding all solutions as well as for finding the first solution. For example, we found without backtracking all solutions to 65 perfect square instances of order 22-25 and sizes ranging from 192×192 to 661×661.

1 Introduction

The utility of *cumulative* constraints in the context of non-overlapping rectangles has been advocated for 15 years in the context of constraint programming [1]. The two main reasons for this utility are: first, it allows to come up with necessary conditions for *non-overlapping* which reuse classical filtering algorithms for *cumulative* like task intervals [2, 3] and compulsory parts [4]; second, it reduces in practice the combinatorial aspect by dividing by a factor of two the number of decision variables of the problem.[1] More recently, *cumulative* constraints have been used by OR researchers [7] in the context of rectangle packing problems for the reasons we have just mentioned. Knapsack constraints were also used, by both OR [8, 7] and CP [9] researchers, to solve the subset-sum problem in the context of scheduling and packing.

In the context of tight rectangle placement problems one can observe that standard filtering methods for the *cumulative* constraint are in fact rather weak. A first reason is that they do not fully take into account the slack (i.e., the difference between the available place and the total area of the rectangles to place) within the filtering process. A second reason is that they relax too much the *cumulative* constraint by allowing to split the tasks into small unit squares. Based on these observations, we decided to

[1] Experiments on tight placement problems have shown [1, 5] that, once all coordinates of the rectangles in one dimension are fixed, it is usually straightforward to extend the partial solution to a full solution even if there exist examples [6] where this is not possible at all.

develop new filtering methods that consider the slack and/or the fact that tasks should not be split into small pieces.

The article is organized as follows. Section 2 recalls the definitions of the $cumulative$ and the *non-overlapping* constraints. Section 3 introduces the longest closed and open hole problems, shows their use in the context of the $cumulative$ and the *non-overlapping* constraints, and provides both bounds and exact methods for these problems. Section 4 presents necessary conditions of the $cumulative$ constraint which consider the available slack, based on knapsack constraints. Section 5 evaluates the contribution of the two methods on perfect and non-perfect placement problems. Finally, Section 6 concludes.

2 Background

In order to model scheduling problems where one has to deal with a resource of limited capacity, the constraint:

$$cumulative(T, \epsilon)$$

was introduced in [1]. Here, T is a collection of tasks, and for a task $t \in T$, $t.\mathrm{s}$, $t.\mathrm{d}$ and $t.\mathrm{h}$ denote respectively its start, duration and height. They all correspond to integer variables[2], while $\epsilon \in \mathbb{Z}^+$ is the height of the resource. The constraint is equivalent to finding an assignment $s : T.\mathrm{s} \rightarrow \mathbb{Z}$ that solves the *cumulative placement* of T of maximum height ϵ, i.e.:

$$\forall i \in \mathbb{Z} : \sigma_s(i) = \epsilon - P(T, i) \geq 0$$

where the *coverage* $P(T, i)$ by T of instant $i \in \mathbb{Z}$ is:

$$P(T, i) = \sum_{t \in T | t.\mathrm{s} \leq i < t.\mathrm{s} + t.\mathrm{d}} t.\mathrm{h}$$

In the context of a $cumulative$ constraint, the *compulsory part* [4] of a task t is the intersection of all feasible instances of t. It can be computed by making the intersection between the task positioned at its earliest start and the task positioned at its latest start. Then the *compulsory part profile* is the aggregation of all compulsory parts of the different tasks of a $cumulative$ constraint. When all tasks that have a non-empty compulsory part are completely fixed, the compulsory part profile is simply called the *cumulative profile*.

In order to handle multi-dimensional placement problems, the constraint $diffn(B)$ was introduced in [10]. Here, B is a collection of boxes, and for a box $b \in B$, $b.\mathrm{s}_k$ and $b.\mathrm{d}_k$ $(1 \leq k \leq n)$ are integer variables that respectively denote the origin and size of b in dimension k. The constraint holds when, for each pair of boxes b, b', there exists at least one dimension k where their projections do not overlap:

[2] An *integer variable* V ranges over a finite set of integers denoted by $\mathcal{D}(V)$. The extremal values in $\mathcal{D}(V)$ are denoted by $\underline{V}$ and $\overline{V}$.

$$\forall b, b' \in B\ (b \neq b'), \exists k \in [1, n] \mid b.\mathrm{s}_k \geq b'.\mathrm{s}_k + b'.\mathrm{d}_k \vee b'.\mathrm{s}_k \geq b.\mathrm{s}_k + b.\mathrm{d}_k$$

As a generalization of *diffn* for handling in a generic way a variety of geometrical restrictions in space and time between polymorphic k-dimensional objects, we proposed in [11] a generic constraint *geost*, whose filtering algorithm was based on a lexicographic sweep algorithm. The methods presented in this article were developed as enhancements to the *geost* constraint.

In the context of this article we focus on the two-dimensional case ($n = 2$), and assume that all the rectangle sizes are fixed. However note that most of the results of this article can be used when we have more than two dimensions, as it is actually the case for our current implementation.

3 The Longest Closed and Open Hole Problems

This section first introduces the *longest closed hole* as well as the *longest open hole* problems and then shows how they can be used in the context of a non-overlapping constraint. Then, it provides both approximate and exact methods for evaluating the longest open and closed holes.

3.1 Defining the Longest Closed and Open Holes

Given a quantity $\sigma \in \mathbb{Z}^+$ of *slack*, the *longest closed hole problem* is to find the largest integer $lcmax_\sigma^\epsilon(T)$ for which there exists a cumulative placement s of a subset of tasks $T' \subseteq T$ of maximum height ϵ, such that the resource area that is not occupied by s on interval $[0, lcmax_\sigma^\epsilon)$ does not exceed the maximum allowed slack value σ:

$$\sum_{i=0}^{lcmax_\sigma^\epsilon - 1} \sigma_s(i) \leq \sigma.$$

The *longest open hole problem* is to find the largest integer $lmax_\sigma^\epsilon(T)$ for which there exist a cumulative placement s of a subset of tasks $T' \subseteq T$ of maximum height ϵ and an interval $[i', i' + lmax_\sigma^\epsilon) \subset \mathbb{Z}$ of length $lmax_\sigma^\epsilon$, such that the resource area that is not occupied by s on $[i', i' + lmax_\sigma^\epsilon)$ does not exceed the maximum allowed slack value σ:

$$\sum_{i=i'}^{i' + lmax_\sigma^\epsilon - 1} \sigma_s(i) \leq \sigma.$$

Example 1. First, consider seven tasks of respective size 11×11, 9×9, 8×8, 7×7, 6×6, 4×4, 2×2. Part (A) of Figure 1 provides a cumulative profile corresponding to the longest open hole problem according to $\epsilon = 11$ and $\sigma = 0$. The longest open hole $lmax_0^{11}(\{11 \times 11, 9 \times 9, 8 \times 8, 7 \times 7, 6 \times 6, 4 \times 4, 2 \times 2\}) = 17$ since:

- The task 8×8 can not contribute since a gap of 3 cannot be filled by the unique candidate the task 2×2.

– The task 6×6 can also not contribute since a gap of 5 cannot be completely filled by the candidates 4×4 and 2×2.

The longest close hole $lcmax_0^{11}(\{11 \times 11, 9 \times 9, 8 \times 8, 7 \times 7, 6 \times 6, 4 \times 4, 2 \times 2\}) = 15$: it corresponds to the longest time interval on which the resource is saturated by the illustrated placement and such that one bound of the interval does not intersect any tasks.

Second, consider a task of size 3×2. Part (B) of Figure 1 provides a cumulative profile corresponding to the longest open hole problem according to $\epsilon = 11$ and $\sigma = 20$. The longest open hole $lmax_{20}^{11}(\{3 \times 2\}) = 2$. □

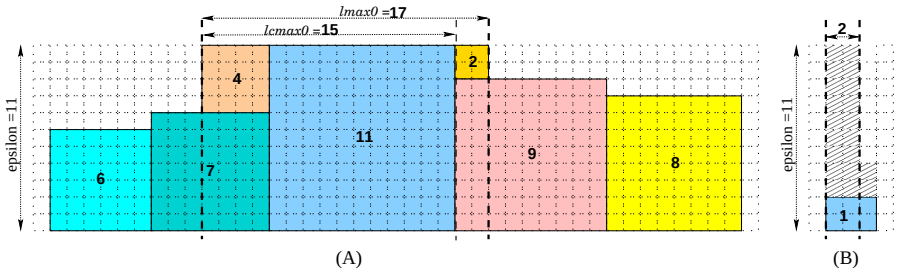

Fig. 1. Examples for illustrating the longest closed and open hole problems

The longest hole problem is a scheduling problem on one cumulative resource with the objective to maximize the period during which the resource is (almost) saturated. To our best knowledge, this precise objective has never been considered in the context of scheduling. The usual goal in scheduling or strip packing problems is to minimize the waste when placing all the given items. The goal here is rather to place a maximum number of items given an allowed amount of waste. When no waste is allowed, that is if $\sigma = 0$, a related problem in the context of two-dimensional packing is called the *longest flat surface*.[3]

Note that a general instance of the longest hole problem (T, ϵ, σ) can easily be transformed to an instance $(T \cup \{1 \times 1\}_\sigma, \epsilon, 0)$ of the same optimum value with no allowed slack and with σ additional unit square dummy tasks.

When $\sigma = 0$ and the height of each task is 1, the decision variant of the longest closed hole problem is to find a partition of the tasks into ϵ disjoint subsets such that the total length of the tasks in each subset is at least a given integer L. Similarly to the bin packing problem (where the subset sums are at most L), this problem has 3-PARTITION as a special case with $|T| = 3\epsilon$, $\frac{L}{4} < t.\mathrm{d} < \frac{L}{2}$, $\forall t \in T$, and $\sum_{t \in T} t.\mathrm{d} = L\epsilon$. Hence it is NP-complete in the strong sense [12].

Thereafter, we present a branch-and-bound to solve the longest hole problems in the open case and in the close case, but we consider also polynomial-time algorithms to solve easy instances of these problems or to compute upper bounds. First, we show in the next section how solutions or bounds of the longest hole problems can be used for filtering the two-dimensional non-overlapping constraint.

[3] see `http://www.stetson.edu/%7Efriedma/mathmagic/1099.html`

3.2 Using the Longest Open and Closed Holes for Filtering

An instance of the $cumulative$ constraint can be mapped into instances of the longest hole problems, where the slack σ is given as the difference between the available and the needed space:

$$\sigma = (\max_{t \in T}(\overline{t.\mathrm{s}} + t.\mathrm{d}) - \min_{t \in T}(\underline{t.\mathrm{s}})) \cdot \epsilon - \sum_{t \in T} t.\mathrm{d} \cdot t.\mathrm{h}.$$

Under this mapping, the longest hole problems can be used in quite a number of different ways in the context of a $cumulative$ constraint, and then in the context of a two-dimensional non-overlapping constraint:

- First, as depicted by Part (A) of Figure 2, it can be used for making an initial pruning of the origin coordinates of the rectangles in order to avoid creating too small holes that cannot be filled enough, with respect to the slack σ, between the border of a rectangle $R1$ and the border of the placement space. For instance, Part (A) illustrates the fact that if, for a given distance $\epsilon \in \mathbb{Z}^+$ between the lower border of a rectangle to place and the lower border of the placement space, the quantity $lmax^{\epsilon}_{\sigma}(\mathcal{R})$ is strictly less than the width of $R1$, then $R1$ cannot start at the corresponding position. $\mathcal{R}$ corresponds to the set of rectangles for which the height does not exceed ϵ (i.e., the rectangles that can fit within the gap). Finally, doing an initial pruning of the origins of the rectangles is important for the knapsack constraints that will be presented in Section 4.
- Second, while fixing both origin coordinates of a rectangle $R1$ during the search, it can also be used to check that the vis-à-vis between $R1$ and each rectangle that is already completely fixed[4] can be filled enough with respect to σ. This is illustrated by Part (B) of Figure 2.
- Finally, it can also be directly used within the two $cumulative$ constraints, which are well-known necessary conditions for a non-overlapping constraint. For this purpose, consider a step of the compulsory part profile (1) that does not reach the resource capacity (i.e., the difference between the resource capacity and the height of the peak is equal to a strictly positive integer ϵ) and (2) that consists of completely fixed tasks. Again, we can use the longest open hole problem in order to check that we can fill enough the gap on top of the given step. This is illustrated by Part (C) of Figure 2.

3.3 Approximations for the Longest Closed and Open Holes

Bounds for the Longest Closed Hole. Without loss of generality, all results, except the termination case, that will be presented later on for the longest open hole apply also for the longest closed hole. This section presents the termination case for the closed hole. Then it considers an upper bound that takes into account the fact that one of the sides of the hole is closed. But first, we introduce a relaxation of the longest closed hole problem where preemption in the cumulative placement is allowed. This relaxation will be used in the proof of the termination case.

[4] Two fixed rectangles have a *vis-à-vis* if and only if (1) they intersect in one dimension, and (2) there is a non-empty gap between them in the other dimension.

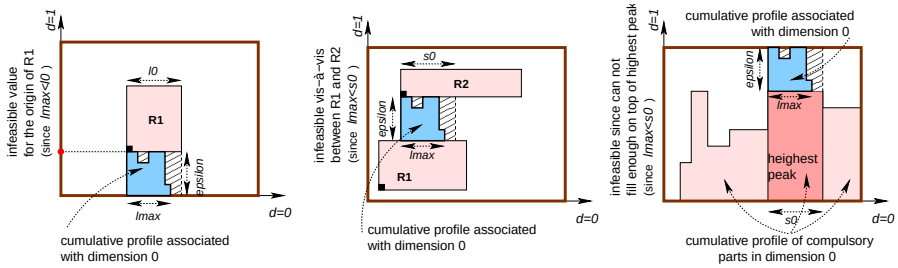

Fig. 2. Three ways of using the longest open hole for filtering a two-dimensional non-overlapping constraint (if the interval is directly located at one border of the placement space we can rather use the longest closed hole).

Preemptive cumulative placement. Allowing preemptive cumulative placements is similar to splitting each task $t \in T$ into a set of $t.\mathrm{d}$ tasks of unit length and of height $t.\mathrm{h}$. Let $\bar{T}$ denote the set of unit length tasks issuing from T. The problem of finding the length $lcmax^{\epsilon}_{\sigma}(\bar{T})$ of the longest closed hole on $\bar{T}$ is a relaxation of the initial problem on T, hence $lcmax^{\epsilon}_{\sigma}(\bar{T})$ is an upper bound of $lcmax^{\epsilon}_{\sigma}(T)$.

Given a preemptive cumulative placement of T, i.e. a cumulative placement of $\bar{T}$, a permutation of the sets $\bar{T}_i$ of (unit length) tasks which are executed at any time $i \in \mathbb{Z}^+$ leads to another preemptive cumulative placement. Such permutations define an equivalence relation between preemptive cumulative placements. In each equivalence class, there exists a placement s such that the sequence $(\sigma_s(i))_{i\in\mathbb{Z}^+}$ of the slacks left at each time is non-decreasing. Such a placement has in its class the longest interval $[0, l_s)$ on which the maximum allowed slack value σ is not exceeded: $l_s = \mathrm{argmax}\{\sum_{i=0}^{l-1}\sigma_s(i) \leq \sigma \mid l \in \mathbb{Z}^+\}$. Let $\mathcal{S}$ denote the set of the preemptive cumulative placements s such that $(\sigma_s(i))_{i\in\mathbb{Z}^+}$ is non-decreasing. Since $lcmax^{\epsilon}_{\sigma}(\bar{T})$ is the maximum length l_s over all the cumulative placements of $\bar{T}$, then there is necessarily one $\bar{s} \in \mathcal{S}$ such that $lcmax^{\epsilon}_{\sigma}(\bar{T}) = l_{\bar{s}}$:

Proposition 1. *If there is a placement $\bar{s} \in \mathcal{S}$ which is minimum in the following sense:*

$$\sigma_{\bar{s}}(i) \leq \sigma_s(i) \text{ or } \sigma_{\bar{s}}(i) = \epsilon \quad \forall s \in \mathcal{S}, \forall i \in \mathbb{Z}^+,$$

then $lcmax^{\epsilon}_{\sigma}(\bar{T}) = l_{\bar{s}}$ for any slack value σ.

Proof. Let $[0, d_s)$ denote the interval on which a placement $s \in \mathcal{S}$ is defined, *i.e.* $\sigma_s(i) = \epsilon$ if and only if $i \geq d_s$. If $\bar{s} \in \mathcal{S}$ is a minimum placement, then $d_{\bar{s}} \leq d_s$ for all $s \in \mathcal{S}$, and trivially, $\sum_{i=0}^{l-1}\sigma_{\bar{s}}(i) \leq \sum_{i=0}^{l-1}\sigma_s(i)$, for all $l \in [0, d_{\bar{s}})$. If $l \geq d_{\bar{s}}$, then $\sum_{i=0}^{l-1}\sigma_{\bar{s}}(i) = l\epsilon - \Sigma$ where Σ is the sum of the surfaces of the tasks in $\bar{T}$, then again $\sum_{i=0}^{l-1}\sigma_{\bar{s}}(i) \leq \sum_{i=0}^{l-1}\sigma_s(i)$, for all $s \in \mathcal{S}$. Hence, the cumulative placement $\bar{s}$ is an optimum solution for the preemptive longest closed hole problem. □

This proof of optimality in the preemptive case allows us to exhibit a solution in polynomial time in the following special non-preemptive case.

Termination rule. Given a set of tasks $T = \{t_i \mid 1 \leq i \leq n\}$ (after reducing their length by applying rule Shrinking 1) such that $(t_i.\mathrm{h}, t_i.\mathrm{d})$ is in non-increasing lexicographic order, let $ndisj$ be the largest integer that satisfies $ndisj = 1 \vee t_{ndisj-1}.\mathrm{h} + t_{ndisj}.\mathrm{h} > \epsilon$ and let $T_{disj} = \{t_i \mid 1 \leq i \leq ndisj\}$ be a non-empty subset of T. If the total height of the tasks in $T \setminus T_{disj}$ plus the maximum height of the tasks in T_{disj} (i.e., $t_1.\mathrm{h}$) is at most ϵ, then the quantity $lcmax^{\epsilon}_{\sigma}(T)$ can be directly evaluated by using the construction depicted by Figure 3.[5]

Proof. By construction, this placement satisfies the conditions of proposition 1, and so it is optimum both for the preemptive relaxation and for the non-preemptive longest closed hole problem. □

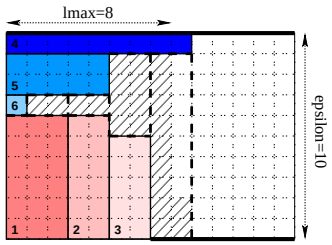

Fig. 3. Illustration of the polynomial case on six tasks 3×6, 2×6, 2×5, 9×1, 5×2 and 1×1 with $\epsilon = 10$ and a slack σ of 23. Tasks $\{3 \times 6, 2 \times 6, 2 \times 5\}$ correspond to disjunctive tasks, while tasks $\{9 \times 1, 5 \times 2, 1 \times 1\}$ correspond to small tasks that can be put on top of the disjunctive tasks. Then the slack $\sigma = 23$ is positioned as early as possible between the disjunctive tasks and the small tasks, which gives a value of 8 for $lcmax^{10}_{23}(\{3 \times 6, 2 \times 6, 2 \times 5, 9 \times 1, 5 \times 2, 1 \times 1\})$.

We now present an upper bound for the longest closed hole problem that is based on the observation that at most one disjunctive task can intersect the rightmost extremity of the interval associated with the longest closed hole.

Upper Bound C1. Given a set of tasks $T = \{t_i \mid 1 \leq i \leq n\}$ (after reducing their length by applying rule Shrinking 1) such that $(t_i.\mathrm{h}, t_i.\mathrm{d})$ is in non-increasing lexicographic order, let $ndisj$ be the largest integer that satisfies $ndisj = 1 \vee t_{ndisj-1}.\mathrm{h} + t_{ndisj}.\mathrm{h} > \epsilon$ and let $T_{disj} = \{t_i \mid 1 \leq i \leq ndisj\}$ be a non-empty subset of T. Moreover let T'_{disj} be the subset of tasks of T_{disj} for which the length was reduced by rule Shrinking 1. If T'_{disj} contains more than one task then let $area\_max_1$ denote the largest area of the tasks of T'_{disj}, and let $area\_rest$ denote the total area of the tasks in $T \setminus T'_{disj}$. We have that $lcmax^{\epsilon}_{\sigma}(T) \leq \lfloor \frac{area\_rest + area\_max_1 + \sigma}{\epsilon} \rfloor$.

[5] Assuming that the tasks were sorted, a direct algorithm implementing this construction runs in $O(n)$ time where n is the number of tasks.

Bounds for the Longest Open Hole. This section shows how to evaluate an upper bound of $lmax^{\epsilon}_{\sigma}(T)$. It assumes that we already know:

- An upper bound of $lmax^{e}_{\sigma}(T)$ for all non-negative integers e that are strictly less than ϵ.
- An upper bound of $lmax^{e}_{\sigma}(T \setminus \{t\})$ for all non-negative integers e that are strictly less than ϵ and for all $t \in T$ for which $t.\mathrm{h} \leq \epsilon$.[6] This quantity will be used for checking what can be put on top of a task t without reusing t.

We first present three rules that simplify the problem by removing some tasks, one rule that reduces the length of some tasks[7], and a rule that computes an exact value of $lmax^{e}_{\sigma}(T)$ when a specific condition on the heights of the tasks holds. Finally, we present two upper bounds of $lmax^{e}_{\sigma}(T)$ and show that they are incomparable.

Simplification 1. Let t be a task of T such that $t.\mathrm{h} > \epsilon$. We have that $lmax^{\epsilon}_{\sigma}(T) = lmax^{\epsilon}_{\sigma}(T \setminus \{t\})$.

Proof. By definition of the longest open hole problem, a task of height strictly greater than ϵ cannot be used. □

Example 2. Consider the set of tasks $T = \{2 \times 2, 4 \times 4, 6 \times 6\}$ and assume that we want to compute $lmax^{1}_{3}(T)$. Using Simplification 1, we have that $lmax^{1}_{3}(T) = lmax^{1}_{3}(\emptyset)$, which means that we can only use the slack of 3 to cover a gap of height 1. Consequently, $lmax^{1}_{3}(T) = 3$. □

Simplification 2. Let T_{ϵ} denote the set of tasks of T for which the heights are equal to ϵ. We have that $lmax^{\epsilon}_{\sigma}(T) \leq \sum_{t \in T_{\epsilon}} t.\mathrm{d} + lmax^{\epsilon}_{\sigma}(T \setminus T_{\epsilon})$.

Example 3. Consider the set of tasks $T = \{2 \times 2, 4 \times 4, 6 \times 6\}$ and assume that we want to compute $lmax^{6}_{0}(T)$. Using Simplification 2, we have that $lmax^{6}_{0}(T) \leq 6 + lmax^{6}_{0}(T \setminus \{6 \times 6\})$. □

Note that in the context of the closed hole we have an equality (i.e., $lmax^{\epsilon}_{\sigma}(T) = \sum_{t \in T_{\epsilon}} t.\mathrm{d} + lmax^{\epsilon}_{\sigma}(T \setminus T_{\epsilon})$).

Simplification 3. Let t be a task of T such that $t.\mathrm{h} < \epsilon$ and $lmax^{\epsilon - t.\mathrm{h}}_{\sigma}(T \setminus \{t\}) = 0$. We have that $lmax^{\epsilon}_{\sigma}(T) = lmax^{\epsilon}_{\sigma}(T \setminus \{t\})$.

Proof. When $lmax^{\epsilon - t.\mathrm{h}}_{\sigma}(T \setminus \{t\})$ is equal to 0, this means that no gap of height $\epsilon - t.\mathrm{h}$ can be filled by the tasks of $T \setminus \{t\}$ without creating an empty space greater than the slack σ. Consequently, if we use task t, we cannot fill enough any gap on top of task t. □

Example 4. Consider the set of tasks $T = \{2 \times 2, 3 \times 3\}$ and assume that we want to compute $lmax^{3}_{0}(T)$. Assume that we already know that $lmax^{1}_{0}(T \setminus \{2 \times 2\}) = 0$. Then, we have that $lmax^{3}_{0}(T) = lmax^{3}_{0}(T \setminus \{2 \times 2\})$. In other words, we can eliminate task 2×2, since we cannot cover any gap of height 1 on top of task 2×2. □

[6] If we don't want to explicitly evaluate an upper bound of $lmax^{e}_{\sigma}(T \setminus \{t\})$, we can take advantage of the fact that $lmax^{e}_{\sigma}(T)$ is an upper bound of $lmax^{e}_{\sigma}(T \setminus \{t\})$.

[7] If, as we will see later, a task cannot contribute with its full length to the longest open hole.

Shrinking 1. Consider a task t of T such that $t.\mathrm{h} < \epsilon$, $t.\mathrm{d} > lmax_\sigma^{\epsilon - t.\mathrm{h}}(T \setminus \{t\})$ and $lmax_\sigma^{\epsilon - t.\mathrm{h}}(T \setminus \{t\}) > 0$. We have that $lmax_\sigma^\epsilon(T) \leq lmax_\sigma^\epsilon(T \setminus \{t\} \cup \{lmax_\sigma^{\epsilon - t.\mathrm{h}}(T \setminus \{t\}) \times t.\mathrm{h}\})$.

Proof. Similar to Simplification 3. We have an inequality since reducing the lengths of more than two disjunctive tasks (i.e., two tasks for which the total height is strictly greater than ϵ) can lead to an overestimation of $lmax_\sigma^\epsilon(T)$. This stems from the fact that at most two disjunctive tasks can be reduced (and the other disjunctive tasks have to be discarded since they would have to be completely included within the interval corresponding to the longest open hole). □

Example 5. Consider the set of tasks $T = \{2 \times 2, 4 \times 4, 6 \times 6\}$ and assume that we want to compute $lmax_0^6(T)$. Suppose we already know that $lmax_0^2(T \setminus \{4 \times 4\}) = 2$. Then we have that $lmax_0^6(T) \leq lmax_0^6((T \setminus \{4 \times 4\}) \cup \{2 \times 4\})$. In other words, the length of task 4×4 is reduced to 2 (i.e., its maximum intersection in time with the longest open hole cannot exceed 2) since, for a gap of 2, we can cover at most a length of 2 without exceeding the slack $\sigma = 0$. □

In the following, all simplification and shrinking rules previously presented are systematically tried before applying the next rule and before evaluating any upper bound.

Termination rule. Given a set of tasks $T = \{t_i \mid 1 \leq i \leq n\}$ (after reducing their length by applying rule Shrinking 1) such that $(t_i.\mathrm{h}, t_i.\mathrm{d})$ is in non-increasing lexicographic order, let:

- $T_{disj_1} = \{t_i \mid 1 \leq i \leq ndisj_1\}$ where $ndisj_1$ is the largest integer strictly less than n that satisfies $t_{ndisj_1}.\mathrm{h} + t_n.\mathrm{h} > \epsilon$, be a subset of T. By construction, these tasks cannot be put in parallel with any task of T.
- $T_{disj_2} = \{t_i \mid ndisj_1 < i \leq ndisj_2\}$ where $ndisj_2$ is the largest integer strictly greater than $ndisj_1$ that satisfies $ndisj_2 = ndisj_1 + 1 \vee t_{ndisj_2 - 1}.\mathrm{h} + t_{ndisj_2}.\mathrm{h} > \epsilon$, be a subset of T. Note that no pair of tasks of T_{disj_2} can be scheduled in parallel.
- $T_{small} = \{t_i \mid ndisj_2 < i \leq n\}$.

If the total height of the tasks in T_{small} plus the maximum height of the tasks in T_{disj_2} is less than or equal to ϵ, then the quantity $lmax_\sigma^\epsilon(T)$ can be directly evaluated by using the construction depicted by Figure 4. [8]

Proof. As in Proposition 1, we consider the preemptive relaxation of the longest open hole problem. It is clear that the construction depicted by Figure 4 is a permutation of an optimum preemptive solution $\bar{s}$ that is minimum among all placements s such that $(\sigma_s(i))_{i \in \mathbb{Z}^+}$ is non-decreasing. □

The intuition of the first upper bound is to consider the total area of the tasks as well as the slack. However, to get a sharper bound we take into account the fact that at most two disjunctive tasks can partially overlap a given interval.

[8] Assuming that the tasks were sorted, a direct algorithm implementing this construction runs in $O(n)$ time where n is the number of tasks.

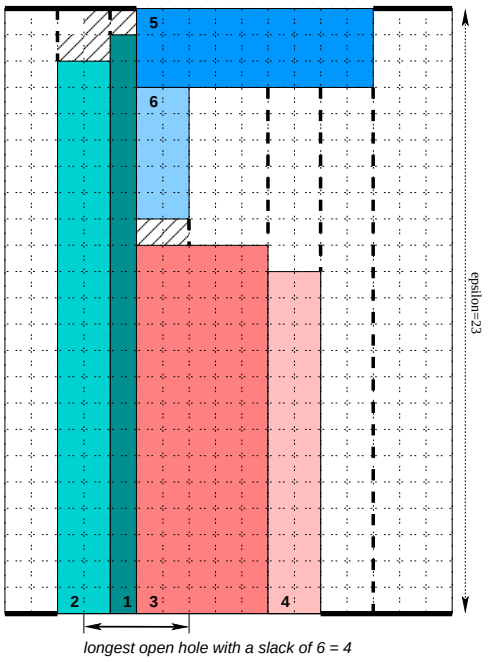

Fig. 4. Illustration of the polynomial case for computing the longest open hole on six tasks 1×22, 2×21, 5×14, 2×13, 2×5 and 9×3 with $\epsilon = 23$ and a slack σ of 6. Tasks $T_{disj_1} = \{1 \times 22, 2 \times 21\}$ correspond to disjunctive tasks that cannot be overlapped by any task, tasks $T_{disj_2} = \{5 \times 14, 2 \times 13\}$ correspond to disjunctive tasks that can only be overlapped by the small tasks, while tasks $T_{small} = \{2 \times 5, 9 \times 3\}$ correspond to small tasks. The slack $\sigma = 6$ is allocated by increasing gap height: using 3 units of slack we first fill the two gaps of height 1. Then we put the remaining slack $6 - 3$ within the gap of height 2, which leads to a longest open hole of size 4.

Upper Bound O1. Given a set of tasks $T = \{t_i \mid 1 \leq i \leq n\}$ (after reducing their length by applying rule Shrinking 1) such that $(t_i.\mathrm{h}, t_i.\mathrm{d})$ is in non-increasing lexicographic order, let $ndisj$ be the largest integer that satisfies $ndisj = 1 \vee t_{ndisj-1}.\mathrm{h} + t_{ndisj}.\mathrm{h} > \epsilon$ and let $T_{disj} = \{t_i \mid 1 \leq i \leq ndisj\}$ be a non-empty subset of T. Moreover, let T'_{disj} be the subset of tasks of T_{disj} for which the lengths were reduced by rule Shrinking 1. If T'_{disj} contains more than two tasks then let $area\_max_1$ and $area\_max_2$ respectively denote the two largest areas of the tasks of T'_{disj}, and let $area\_rest$ denote the total area of the tasks in $T \setminus T'_{disj}$. We have that $lmax^{\epsilon}_{\sigma}(T) \leq \lfloor \frac{area\_rest + area\_max_1 + area\_max_2 + \sigma}{\epsilon} \rfloor$.

Proof. Given a fixed interval $[low, up]$ and a set of disjunctive tasks T_{disj}, at most two tasks of T_{disj} can partially overlap interval $[low, up]$. Note that disjunctive tasks for which the lengths were reduced cannot be completely included within interval $[low, up]$ (i.e., they either overlap one border of interval $[low, up]$, or they don't overlap at all interval $[low, up]$). Consequently, if we reason in terms of areas, we can only consider the two largest areas of the disjunctive tasks for which the lengths were reduced. □

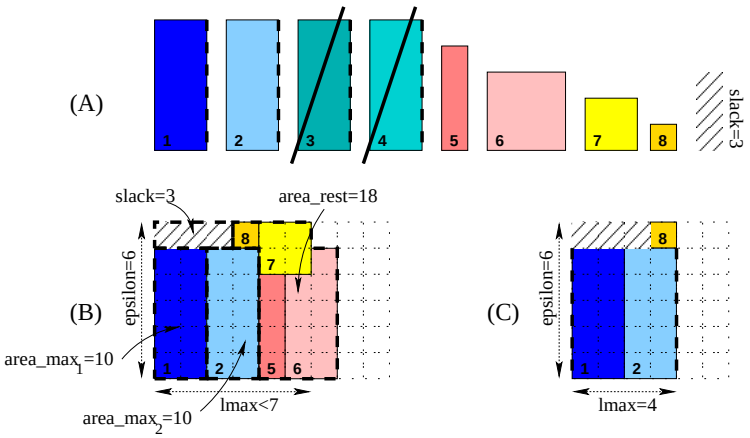

Fig. 5. (B) Illustration of the first upper bound on eight tasks (A) $T = \{2 \times 5, 2 \times 5, 2 \times 5, 2 \times 5, 1 \times 4, 3 \times 3, 2 \times 2, 1 \times 1\}$, where the length of the first four tasks was reduced by applying rule Shrinking 1 (this reduction is depicted by a dashed line along the right border of a task), with $\epsilon = 6$ and a slack σ of 3. (C) A placement giving the exact value for $lmax^6_3(T)$. Note that a task for which the length was reduced can only be put at one of the two extremities of the placement; consequently, we cannot add task 5 and task 7 to gain an extra unit for $lmax^6_3(T)$ (since the lengths of tasks 1 and 2 were reduced, tasks 1 and 2 have to be kept at one of the two extremities).

Example 6. Figure 5 illustrates the computation of the first upper bound. From the set of tasks T we can construct the set $T_{disj} = \{2 \times 5, 2 \times 5, 2 \times 5, 2 \times 5, 1 \times 4, 3 \times 3\}$ of disjunctive tasks, since for any pair of tasks in T_{disj} we have that their total height exceeds $\epsilon = 6$. By hypothesis, $T'_{disj} = \{2 \times 5, 2 \times 5, 2 \times 5, 2 \times 5\}$ and $area\_max_1 = area\_max_2 = 10$. Finally, the total area of the tasks in $T \setminus T'_{disj}$, $area\_rest$, is equal to $4 + 9 + 4 + 1 = 18$. Consequently, $lmax^6_3(T) \leq \lfloor \frac{18+10+10+3}{6} \rfloor = 6$. □

The intuition of the next upper bound is not to reason any more just in terms of area, but to take into account the fact that disjunctive tasks cannot be put in parallel. We sort the disjunctive tasks by decreasing height and try now to reduce their length according to the tasks (and the slack) that can be effectively placed on top of the disjunctive tasks.

Upper Bound O2. Given a set of tasks $T = \{t_i \mid 1 \leq i \leq n\}$ (after reducing their length by applying rule Shrinking 1) such that $(t_i.\mathrm{h}, t_i.\mathrm{d})$ is in non-increasing lexicographic order, let $ndisj$ be the largest integer that satisfies $ndisj = 1 \vee t_{ndisj-1}.\mathrm{h} + t_{ndisj}.\mathrm{h} > \epsilon$ and let $T_{disj} = \{t_i \mid 1 \leq i \leq ndisj\}$ be a non-empty subset of T. For any h, let $area_h$ denote the total area of the tasks of T that have a height less than or equal to h. If there is an $i \in [1, ndisj]$ such that:

- $\forall j \in [1, i-1], \sum_{k=1}^{j} t_k.\mathrm{d} \cdot t_k.\mathrm{h} + area_{\epsilon - t_j.\mathrm{h}} + \sigma \geq \sum_{k=1}^{j} t_k.\mathrm{d} \cdot \epsilon$,
- $\sum_{k=1}^{i} t_k.\mathrm{d} \cdot t_k.\mathrm{h} + area_{\epsilon - t_i.\mathrm{h}} + \sigma < \sum_{k=1}^{i} t_k.\mathrm{d} \cdot \epsilon$,

then the length of task t_i can be reduced to $\lfloor \frac{area_{\epsilon - t_i.\mathrm{h}} + \sigma - \sum_{k=1}^{i-1} t_k.\mathrm{d} \cdot (\epsilon - t_k.\mathrm{h})}{\epsilon - t_i.\mathrm{h}} \rfloor$.

Now let T'_{disj} be the set of tasks derived from T_{disj} by considering their reduced lengths, discarding tasks of reduced length equal to 0. Let $area = \sum_{t \in T \setminus T'_{disj}} t.\mathrm{d} \cdot t.\mathrm{h} + \sigma$ and let $t'_1, t'_2, \ldots, t'_{ndisj'}, t'_{ndisj'+1}$ denote the tasks of T'_{disj} sorted by decreasing height, where $t'_{ndisj'+1}$ stands for an additional task of height 0 and length $\lceil \frac{area}{\epsilon} \rceil$. In this context, let i be the smallest integer such that $\sum_{k=1}^{i} t'_k.\mathrm{d} \cdot (\epsilon - t'_k.\mathrm{h}) \geq area$. We have that $lmax_\sigma^\epsilon(T) \leq \sum_{k=1}^{i-1} t'_k.\mathrm{d} + \lfloor \frac{area - \sum_{k=1}^{i-1} t'_k.\mathrm{d} \cdot t'_k.\mathrm{h}}{\epsilon - t'_i.\mathrm{h}} \rfloor$.

Example 7. Figure 6 provides an example of application of the second upper bound on a set of tasks $T = \{3 \times 5, 2 \times 4, 2 \times 4, 5 \times 3, 3 \times 3, 2 \times 2, 1 \times 1\}$ under the hypothesis that we have a slack $\sigma = 3$ and a gap $\epsilon = 6$. The set of disjunctive tasks T_{disj} built from these rectangles is $\{3 \times 5, 2 \times 4, 2 \times 4, 5 \times 3\}$. The length of task t_3 (i.e., $i = 3$) can be reduced since:

- $[j = 1]$: $t_1.\mathrm{d} \cdot t_1.\mathrm{h} + area_{6-5} + \sigma = 3 \cdot 5 + 1 + 3 = 19 \geq 3 \cdot 6$,
- $[j = 2]$: $t_1.\mathrm{d} \cdot t_1.\mathrm{h} + t_2.\mathrm{d} \cdot t_2.\mathrm{h} + area_{6-4} + \sigma = 3 \cdot 5 + 2 \cdot 4 + 5 + 3 = 31 \geq (3+2) \cdot 6$,
- $[i = 3]$: $t_1.\mathrm{d} \cdot t_1.\mathrm{h} + t_2.\mathrm{d} \cdot t_2.\mathrm{h} + t_3.\mathrm{d} \cdot t_3.\mathrm{h} + area_{6-4} + \sigma = 3 \cdot 5 + 2 \cdot 4 + 2 \cdot 4 + 5 + 3 = 39 < (3+2+2) \cdot 6$.

The length of task $t_3 = 3 \times 4$ is reduced to $\lfloor \frac{area_{\epsilon - t_3.\mathrm{h}} + \sigma - t_1.\mathrm{d} \cdot (\epsilon - t_1.\mathrm{h}) - t_2.\mathrm{d} \cdot (\epsilon - t_2.\mathrm{h})}{\epsilon - t_3.\mathrm{h}} \rfloor = \lfloor \frac{5+3-3 \cdot (6-5) - 2 \cdot (6-4)}{6-4} \rfloor = 0$. Consequently, $lmax_3^6(T) = 8$ (instead of 9 if t_3 is not removed). □

This second upper bound can be enhanced by trying to compute a bigger list of tasks in disjunction. A task t_i cannot overlap a task t_j if their total height, $t_i.\mathrm{h} + t_j.\mathrm{h}$, is greater than ϵ. But we can also use the fact that we have already computed the longest open hole for smaller values of ϵ. Tasks t_i and t_j are also in disjunction if there is a gap g on top of the two tasks (i.e., $g = \epsilon - t_i.\mathrm{h} - t_j.\mathrm{h}$) for which $lmax_\sigma^g(T \setminus \{t_i, t_j\})$ is equal to 0.

Illustrating the Incomparability of the Two Bounds. This paragraph shows that the two bounds previously described are in fact incomparable. For this purpose, consider the tasks of size 2×2, 4×4, 6×6, 7×7, 8×8, 9×9, 11×11 and 15×15. Let $\mathrm{B1}_\sigma^\epsilon$ and $\mathrm{B2}_\sigma^\epsilon$ respectively denote the upper bounds for $lmax_\sigma^\epsilon$ obtained by the first and the second upper bounds previously introduced. On the one hand, we have that $\mathrm{B1}_0^{12} = 5$ and $\mathrm{B2}_0^{12} = 4$, while on the other hand we have that $\mathrm{B1}_0^{15} = 30$ and $\mathrm{B2}_0^{15} = 32$.

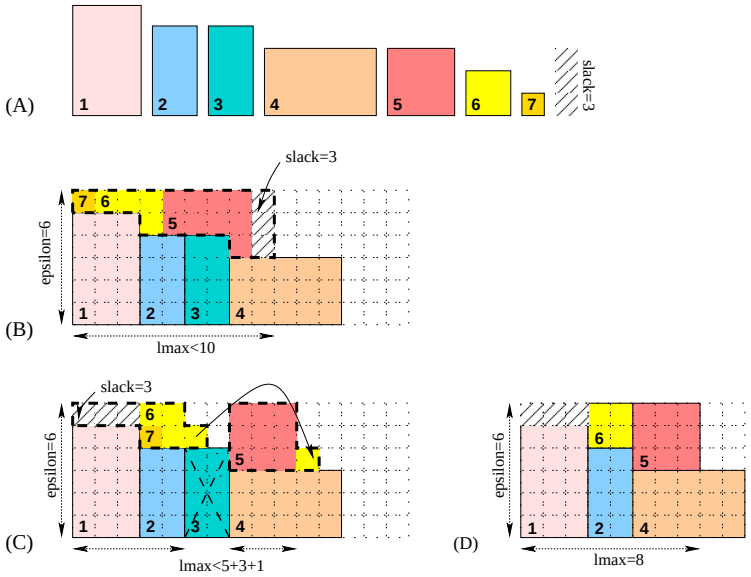

Fig. 6. (A) Seven tasks 3×5, 2×4, 2×4, 5×3, 3×3, 2×2 and 1×1 to place with a slack of 3 and a gap ϵ of 6, (B) An upper bound of 9 obtained without shrinking, (C) A tighter upper bound of 8 obtained by removing the third task, (D) An optimal placement which reaches the bound 8.

3.4 Exact Methods for the Longest Closed and Open Holes

Early experiments with *geost* have shown that the current bounds usually provide optimal solutions when the number of tasks is small (i.e., does not exceed 7). However for a bigger number of tasks the gap between the upper bound and the optimal value increases quite rapidly, especially in the context of the longest closed hole. Figures 7 and 8 respectively provide the exact values for n consecutive squares both for the closed and the open cases when (1) both values coincide and when (2) the exact value associated with the closed case is strictly less than the exact value of the open case.

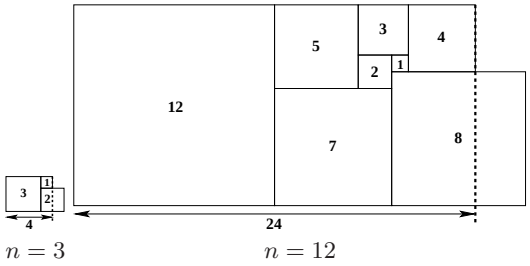

Fig. 7. Exact values for the longest closed and open hole for n distinct squares of sizes $1, 2, \ldots, n$ $(n \in \{3, 12\})$ with zero slack and a gap ϵ of size 3 and of size 12

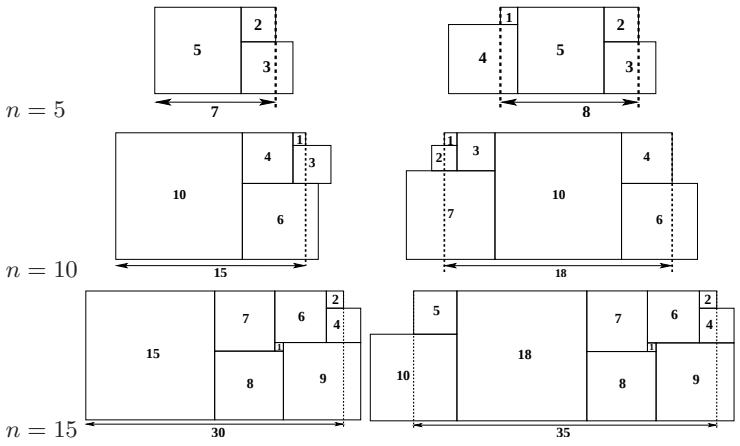

Fig. 8. Exact values for the longest closed and open hole for n distinct squares of sizes $1, 2, \ldots, n$ ($n \in \{5, 10, 15\}$) with zero slack and a gap ϵ of sizes $5, 10, 15$ where the exact value of the closed (left part) and open (right part) cases do not coincide

In order to investigate how having suboptimal values affects the pruning ability and the whole performance, we have developed an exact branch and bound method for the two longest hole problems. This section presents the exact method for the closed case. Since the open case can be obtained as an easy generalization of the closed case its presentation will be omitted. The actual contributions of these exact methods will be evaluated in Section 5.

Algorithm 1 provides a sketch of the method. Initially all tasks are unscheduled and their respective earliest starts are all set to 0 (see line 4). Then, while the best solution found so far $cmax$ is strictly less than the upper bound $umax$ and the search tree is not exhausted we:

- Determine the earliest start δ among the unscheduled tasks (see line 6).
- Check that we are not in a failure case (see line 9) where we backtrack (see line 10):
  dominated An unscheduled task can be scheduled so that:
    1. Its end is less than or equal to δ.
    2. It does not produce any resource overflow with respect to the capacity ϵ and to the tasks that were already scheduled.

  slack_overflow One on the following condition holds:
    1. The slack $\sigma_{[0,\delta)}$ on interval $[0, \delta)$ on top of the cumulative profile (i.e., the empty space) of the scheduled tasks exceeds the slack σ.
    2. Too much space is lost at instant δ. More precisely, let $sumh_\delta$ be the total height of the unscheduled tasks for which (1) the earliest start is equal to δ, and (2) the height is not greater than $\epsilon - h_\delta$. The quantity $\epsilon - h_\delta - sumh_\delta$ is strictly greater than $\sigma - \sigma_{[0,\delta)}$.

  too_big_loss One on the following condition holds:
    1. There is an interval $[\delta, \delta + \alpha]$ of the cumulative profile for which the gap cannot be filled enough (i.e., we lose more than $\sigma - \sigma_{[0,\delta)}$ slack on top of that interval). Estimating the loss is done by consulting the $lmax$ table, which is computed by increasing values of ϵ.

PROCEDURE LongestClosedHole$(\epsilon, \sigma, n, tasks)$: int

1: $umax \leftarrow$ upper bound for the longest closed hole attached to parameters $\epsilon, \sigma, n, l[], h[]$
2: $cmax \leftarrow 0$ // best solution found so far
3: $stack \leftarrow$ empty // initialize to empty stack of pending choices
4: set earliest start of all tasks to 0 and all tasks to unscheduled
5: **repeat**
6: $\delta \leftarrow$ earliest start of the unscheduled tasks
7: $h_\delta \leftarrow$ total height of the scheduled tasks that cross instant δ
8: $t \leftarrow$ first unscheduled task for which the earliest start is equal to δ
9: **if** dominated $\vee$ slack_overflow $\vee$ too_big_loss $\vee$
resource_overflow(t) $\vee$ dominated(t) **then**
10: pop $stack$
11: **else if** polynomial case **then**
12: $cmax \leftarrow \max(cmax$,polynomial bound)
13: pop $stack$
14: **else**
15: $cmax \leftarrow \max(cmax$,usual bound with the scheduled tasks and the initial slack)
16: schedule task t at instant δ and push t onto $stack$
17: **end if**
18: **until** $cmax \geq umax \ \vee$ empty $stack$
19: **return** $cmax$

Algorithm 1: Exact method for computing the longest closed hole.

2. Since too many tasks were delayed we lose too much space on interval $[\delta, emin - 1]$, where $emin$ is the earliest start of an unscheduled task such that $\delta + 2 \leq emin \leq cmax$. This is done by computing the maximum intersection of each unscheduled task [9] with interval $[\delta, emin - 1]$.

resource_overflow(t) Scheduling the not yet scheduled task t at instant δ causes to exceed the resource capacity ϵ.

dominated(t) The unscheduled task t could be rescheduled at instant $\delta - 1$.

– Check that we are not in a polynomial case (see line 11) where we can directly compute the exact value of the longest closed hole (see line 12) and backtrack (see line 13). The polynomial case corresponds to the termination rule depicted in Figure 3.
– Finally, if we are neither in a failure case, nor in the polynomial case (see line 14) we first update $cmax$ (see line 15) by considering just all the scheduled tasks as well as the initial slack σ (i.e., starting from instant 0 we use the initial slack σ in order to fill the gap on top of the cumulative profile until we have used up σ). Then, we add task t to the profile and put it on the stack of pending choices (see line 16) .

4 Balancing Knapsack Constraints

In the context of a $cumulative(T, \epsilon)$ constraint with $|T| = n$ and slack σ, let its timespan be defined as $[u_{\min}, u_{\max})$ where $u_{\min} = \min_{t \in T} \underline{t.\mathrm{s}}$ and $u_{\max} = \max_{t \in T} \overline{t.\mathrm{s}} + t.\mathrm{d}$.

[9] In order to computing the maximum intersection of an unscheduled task with $[\delta, emin - 1]$ we consider the earliest start of that task.

If $\sigma = 0$, for every time point $b \in [u_{\min}, u_{\max})$, the coverage of b must equal ϵ. If $\sigma > 0$, the total coverage of a given time point must be in $[\epsilon - \sigma, \epsilon]$, while the total coverage of any two time points must be at least $2\epsilon - \sigma$. In particular, we have the following necessary conditions for the constraint:

$$\begin{array}{c} \forall b \in [u_{\min}, u_{\max}) : P(T, b) \in [\epsilon - \sigma, \epsilon] \\ \forall b \in [u_{\min} + 1, u_{\max}) : P(T, b-1) + P(T, b) \in [2\epsilon - \sigma, 2\epsilon] \end{array} \tag{1}$$

For every time point $b \in [u_{\min} + 1, u_{\max})$ and task $t_i \in T$, we have four mutually exclusive possibilities. We encode these possibilities as 0-1 variables $S_{ib}, C_{ib}, O_{ib}, N_{ib}$ where $S_{ib} + C_{ib} + O_{ib} + N_{ib} = 1$ and:

$$\begin{array}{l} S_{ib} = 1 \Leftrightarrow t_i \text{ intersects } b \text{ but not } b-1, \text{ that is, } t_i.\mathrm{s} = b \\ C_{ib} = 1 \Leftrightarrow t_i \text{ intersects } b-1 \text{ but not } b, \text{ that is, } t_i.\mathrm{s} = b - t_i.\mathrm{d} \\ O_{ib} = 1 \Leftrightarrow t_i \text{ intersects both } b-1 \text{ and } b, \text{ that is, } t_i.\mathrm{s} \in [b - t_i.\mathrm{d} + 1, b - 1] \\ N_{ib} = 1 \Leftrightarrow t_i \text{ intersects neither } b-1 \text{ nor } b, \text{ that is, } t_i.\mathrm{s} \notin [b - t_i.\mathrm{d}, b] \end{array} \tag{2}$$

For a given time point b, the set of tasks T and the above, we can set up the following pseudo-boolean equation system, which essentially captures necessary conditions (1).

$$\begin{array}{l} \forall i \in [1, n] : S_{ib} + C_{ib} + O_{ib} + N_{ib} = 1 \\ H_{b-1} = \sum_{i \in [1,n]} t_i.\mathrm{h} \cdot (C_{ib} + O_{ib}) \in [\epsilon - \sigma, \epsilon] \\ H_b = \sum_{i \in [1,n]} t_i.\mathrm{h} \cdot (S_{ib} + O_{ib}) \in [\epsilon - \sigma, \epsilon] \\ H_{b-1} + H_b \in [2\epsilon - \sigma, 2\epsilon] \end{array} \tag{3}$$

This equation system can be solved by a dynamic programming method similar to the one described in [13]. Define a function $f(k, l, r)$ equal to 1 if and only if the derived equation system (4) has a solution, and define the dynamic programming recursion as in (5).

$$\begin{array}{l} \forall i \in [1, k] : S_{ib} + C_{ib} + O_{ib} + N_{ib} = 1 \\ \sum_{i \in [1,k]} t_i.\mathrm{h} \cdot (C_{ib} + O_{ib}) = l \\ \sum_{i \in [1,k]} t_i.\mathrm{h} \cdot (S_{ib} + O_{ib}) = r \end{array} \tag{4}$$

$$\begin{array}{l} f(0, l, r) = \begin{cases} 1 \ , \text{ if } l = 0 \wedge r = 0 \\ 0 \ , \text{ otherwise} \end{cases} \\ f(k, l, r) = \max \begin{cases} f(k-1, l, r - t_k.\mathrm{h}) \\ f(k-1, l - t_k.\mathrm{h}, r) \\ f(k-1, l - t_k.\mathrm{h}, r - t_k.\mathrm{h}) \\ f(k-1, l, r) \end{cases} \text{, if } k > 0 \end{array} \tag{5}$$

Now, intuitively, (3) has a solution if and only if there exist l and r such that $l \in [\epsilon - \sigma, \epsilon]$, $r \in [\epsilon - \sigma, \epsilon]$, $l + r \in [2\epsilon - \sigma, 2\epsilon]$, and $f(n, l, r) = 1$. One can visualize this 3-dimensional knapsack problem as a directed graph with a node for every (k, l, r) for which $f(k, l, r) = 1$ and arcs corresponding to (5). Also, each arc is annotated with the 0-1 variable that takes the value 1 in that branch of the recursion:

$$(k-1, l, r - t_k.\mathrm{h}) \xrightarrow{S_{kb}} (k, l, r)$$
$$(k-1, l - t_k.\mathrm{h}, r) \xrightarrow{C_{kb}} (k, l, r)$$
$$(k-1, l - t_k.\mathrm{h}, r - t_k.\mathrm{h}) \xrightarrow{O_{kb}} (k, l, r)$$
$$(k-1, l, r) \xrightarrow{N_{kb}} (k, l, r)$$

Among the nodes, let the single *source* node be $(0, 0, 0)$, and let the *sink* nodes be all nodes (n, l, r) where $l \in [\epsilon - \sigma, \epsilon]$, $r \in [\epsilon - \sigma, \epsilon]$, and $l + r \in [2\epsilon - \sigma, 2\epsilon]$. Then a path from the source to some sink corresponds to a solution to (3). By inspecting the arcs of such paths, we can determine for each 0-1 variable whether it takes the value 1 in some solution to (3). After computing all paths, we inspect each 0-1 variable: if it does not take the value 1 in any solution, the corresponding start time domain is pruned according to the equivalences given in (2). The complexity of this algorithm is $O(nL^2)$ (space and time).

Example 8. Consider a $cumulative(\{t_1, t_2, t_3\}, 6)$ constraint with the following tasks:

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

and let us apply the method for $b = 4$ and $\sigma = 4$ (the slack has been tightened by other, fixed tasks that have been omitted in the example). The method explores the digraph shown in Figure 9. The four sink nodes are denoted by ellipses. As there is no arc annotated with O_{3b} on a path reaching a sink, we conclude that t_3 cannot intersect both 3 and 4, hence the value 3 can be removed from $\mathcal{D}(t_3.\mathrm{s})$. In this example, the digraph is a tree, which is not generally the case. □

4.1 Strengthening the Method

The method can be strengthened by adding more dimensions to the knapsack problems, e.g., by adding constraints that capture the fact that the height of the cumulative profile must not exceed ϵ. This can be done as follows:

- Identify subsets $T_l \subseteq T$ and $T_r \subseteq T$ such that the following properties hold:
  - For each $t_i \in T_l$, both 0 and 1 are feasible values for O_{ib}, and $O_{ib} = 0$ would create a compulsory part of t_i to the left of b.
  - If $O_{ib} = 0$ for all $t_i \in T_l$, the cumulative profile would exceed ϵ.
  - For each $t_i \in T_r$, both 0 and 1 are feasible values for O_{ib}, and $O_{ib} = 0$ would create a compulsory part of t_i to the right of b.
  - If $O_{ib} = 0$ for all $t_i \in T_r$, the cumulative profile would exceed ϵ.
- Add the constraint $\sum_{t_i \in T_l} O_{ib} \geq 1$ for every such subset T_l found.
- Add the constraint $\sum_{t_i \in T_r} O_{ib} \geq 1$ for every such subset T_r found.

Our implementation includes this idea, using at most one subset T_l and at most one subset T_r.

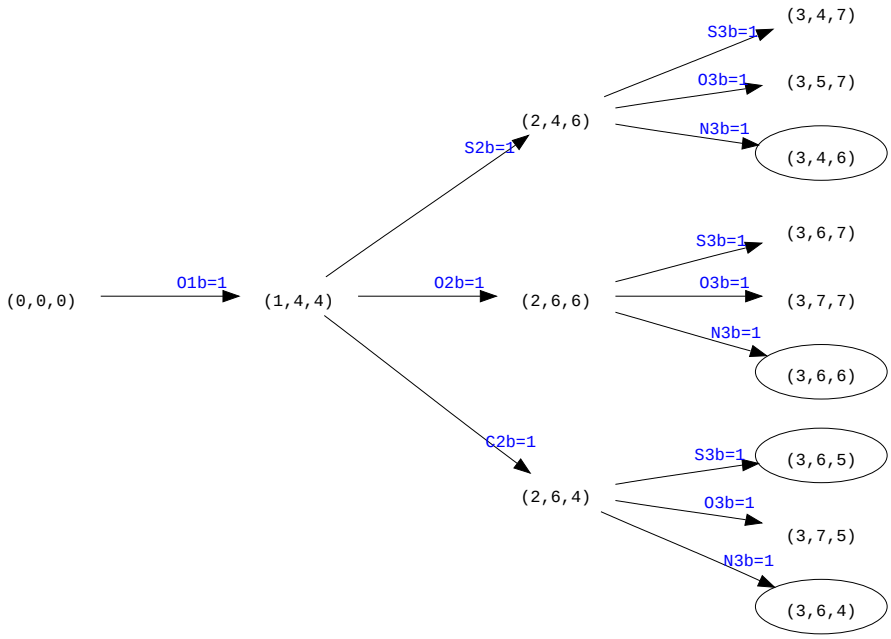

Fig. 9. Tasks and digraph explored by dynamic programming for $b = 4, \sigma = 4, \epsilon = 6$.

4.2 Learning Solutions

Let the *pre-signature* of a $cumulative(T, \epsilon)$ constraint and time point b be the set of 0-1 variables for which the value 1 is feasible according to (2) prior to solving the equation system. Similarly, let its *post-signature* be the set of 0-1 variables for which the value 1 is still feasible after solving the equation system.

It is worth noting that, given fixed T, ϵ, σ, the pseudo-boolean equation system is totally abstracted away from the chosen b as well as from the variable domains. It is totally determined by its pre-signature. Thus having solved an equation system, it makes sense to record its pre- and post-signatures. Later on, if an equation system with the same pre-signature recurs, we can retrieve the associated post-signature instead of recomputing it. Experience shows that this idea saves about 75% of the computational effort.

5 Performance Evaluation

All the new filtering methods described in this article were integrated into our *geost* kernel [11] in order to strengthen the sweep-based filtering associated with non-overlapping constraints. The experiments were run in SICStus Prolog 4 compiled with gcc -02 version 4.1.0 on a 3GHz Pentium IV with 1MB of cache. All benchmarks were run without any symmetry breaking and with the following four-phase search procedure, also known

as *Interval* [14], where at each phase, rectangles are considered by decreasing area. Let $o.x$ denote the X coordinate variable of the rectangle o:

1. For each rectangle 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.[10]
2. For each rectangle o, fix $o.x$ by binary search.
3. Repeat steps 1-2 for the Y coordinates.

To evaluate the effectiveness of the two methods described in this article, we have tried six variants of the *geost* propagator, denoted as follows, where $\sigma_{\max}$ denotes the slack of the problem:

G1 *geost* + balancing knapsack constraints.
G2 *geost* + balancing knapsack constraints + longest hole table for $0 \leq \sigma \leq \sigma_{\max}$ computed using the upper bounds enhanced with the exact method allowing 10000 backtracks.
G3 *geost* only.
G4 *geost* + longest hole table for $0 \leq \sigma \leq \sigma_{\max}$ computed using the upper bounds.
G5 *geost* + longest hole table for $0 \leq \sigma \leq \sigma_{\max}$ computed using the upper bounds enhanced with the exact method allowing 1000 backtracks.
G6 *geost* + longest hole table for $0 \leq \sigma \leq \sigma_{\max}$ computed using the upper bounds enhanced with the exact method allowing 10000 backtracks.

Wanting to get an idea of their performance on perfect packing problems (no slack), we considered the *perfect square problem* [1, 9]. A *perfect square of order* n is a square that can be tiled with n smaller squares where each of the smaller squares has a different integer size. We used the data available (i.e., the size of the small squares to pack) from the catalogue [15] and tested the corresponding 207 instances. In Figure 10, we show the number of problems solved (all solutions) for (a) a given backtrack limit using G2, (b) a given time limit using G5.

In order to evaluate the impact of our two methods and their variants, we solved the 207 instances by the various variants and inspected the amount of backtracks and time needed to solve each instance. Figure 11 shows a table giving the total number of backtracks and time used per method. In our CP'07 paper [11], we reported a total of 13109 seconds and 846416 backtracks to solve all instances, so our new results is an improvement by more than one resp. two orders of magnitude in terms of runtime resp. backtracks. The figure also contains four scatter plots. In each plot, one variant, A, is run vs. another variant, B. Each dot corresponds to a problem instance. If solving a given instance requires x backtracks in variant A and y backtracks in variant B, then a dot is placed at coordinates (x, y). The plot on the upper left hand side shows the effect of the balancing knapsack constraints. The other three plots show the effect of the longest hole table in three variants. First of all, we find that that both methods sharply decrease the number of backtracks, balancing knapsack constraints having the strongest effect (cf. G1, G3 and G4). Secondly, we find a nice multiplicative effect from combining the

[10] Simonis et al. [14] propose instead to split the domain of $o.x$ into intervals of length 0.3 of the rectangle length. We found that our variant produced better results, likely due to our new methods that exploit the compulsory part profile.

two methods (cf. G1, G2 and G6). Thirdly, we find that the tighter longest hole table computed by the exact method leads to stronger filtering (cf. G4 and G5), but also that spending a lot of time in the exact method does not improve the overall runtime (cf. G5 and G6).

In order to evaluate how our methods perform on non-perfect packing problems (with non-zero slack), we took a subset of the same 207 perfect square instances, successively introduced slack into each instance by removing one or more of the smallest squares, and measured the number of backtracks required to find the *first* solution to the non-perfect packing problem thus obtained. The study was limited to problems with at most 5% slack that could be solved in at most 1 CPU minute. Figure 12 summarizes the results. Here we find that as the amount of slack increases, our methods degrade but still have a noticeable effect, at least of the slack does not exceed 5%. However the results should be treated with caution, for these experiments were performed on a single benchmark set, and moreover another search procedure would perhaps show different results. When we tried the methods on the 2D orthogonal packing instances proposed by Clautiaux et al. [7], the two methods did not significantly decrease the number of backtracks on non-perfect packing instances, whereas on instances with zero slack they did, which is somewhat consistent with the perfect square results.[11]

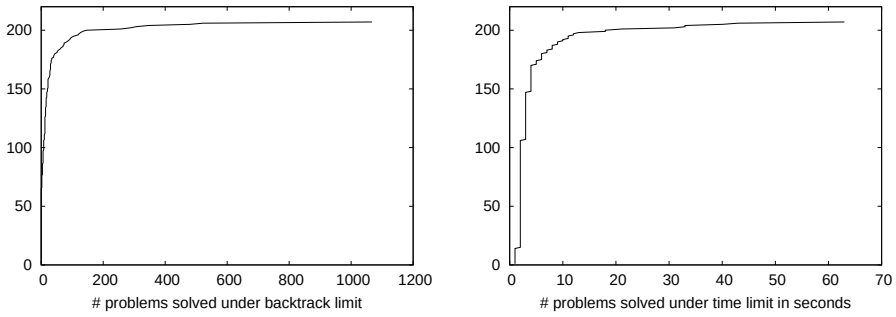

Fig. 10. Perfect packing problems, all solutions. Left: Number of problems solved by G2 under backtrack limit. 65 problems were solved without backtrack. Right: Number of problems solved by G5 under time limit in seconds. 106 problems required less than 2 CPU seconds to solve.

6 Conclusion

This article introduces two new filtering methods that can be used in the context of the $cumulative$ as well as the *non-overlapping* constraints.

[11] Unlike the squares instances, it is worth noting that the Clautiaux instances contain rectangles that are long in one dimension and short in the other dimension.

Version	Total Backtracks	Total Seconds	knapsack	longest hole
G1	21226	2928	yes	no
G2	*6736*	2299	yes	$0 \leq \sigma \leq \sigma_{\max}, \leq 10000$ backtracks
G3	1313279	4652	no	no
G4	261910	1041	no	$0 \leq \sigma \leq \sigma_{\max}$, bounds only
G5	174979	*840*	no	$0 \leq \sigma \leq \sigma_{\max}, \leq 1000$ backtracks
G6	164366	1159	no	$0 \leq \sigma \leq \sigma_{\max}, \leq 10000$ backtracks

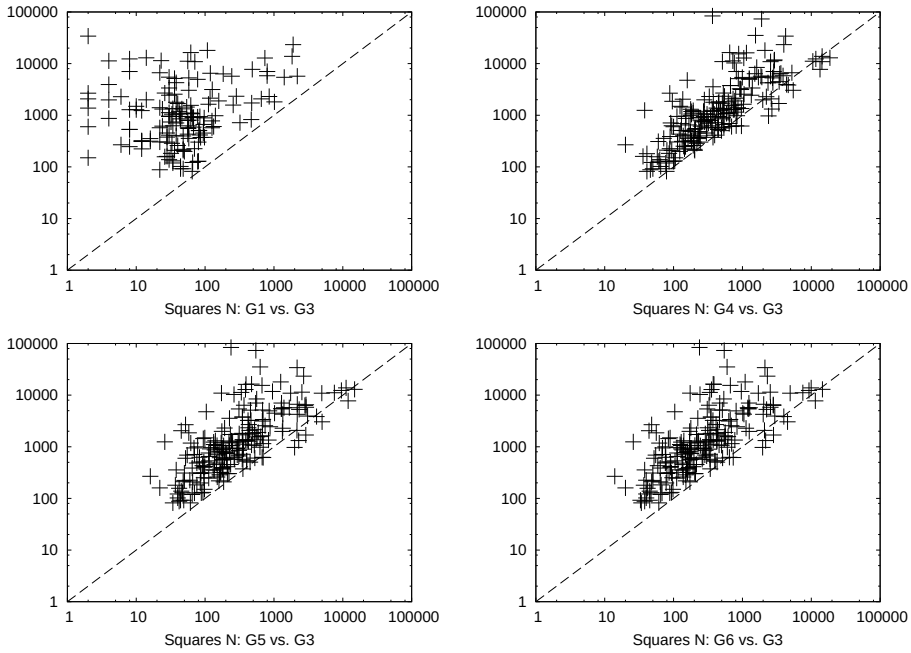

Fig. 11. Perfect packing problems, all solutions. Top: performance summary as total backtracks (seconds) per variant. Bottom: scatter plots of number of backtracks per instance for selected pairs of variants.

σ	# problems	G3	G1	G4
0	134	340(1.000)	22(0.064)	220(0.647)
(0,0.005)	567	685(1.000)	436(0.636)	520(0.759)
[0.005.01)	103	2940(1.000)	2712(0.922)	1906(0.648)
[0.01,0.02)	69	3800(1.000)	3515(0.925)	3124(0.822)
[0.02,0.03)	28	4165(1.000)	3455(0.829)	3556(0.854)
[0.03,0.04)	15	5472(1.000)	5296(0.968)	4506(0.824)
[0.04,0.05)	12	175(1.000)	166(0.948)	175(0.999)

Fig. 12. Non-perfect packing problems, first solution. Variants G1, G3 and G4 are compared. Each row shows: the number of problems with slack in a given interval and the average number of backtracks required to solve a problem. Numbers in parentheses show the improvement wrt. the data given in column **G3**, that is, gives an indication of how effective the respective method is in the presence of the given amount of slack.

1. The *longest open and closed hole problems* can be used to detect that some specific part of the placement space cannot be filled enough.
2. The *balancing knapsack constraints* can be used to detect tasks that must resp. must not cross a given pair of time instants in order to not under- or overfill those instants.

As demonstrated by our benchmarks, these two methods are complementary, especially when the slack is very small. In such contexts, they reduce significantly the number of backtracks and even allow to completely enumerate the search space for a significant number of instances without any backtrack. An open issue is to come up with more efficient methods for proving infeasibility when the slack is not so small.

Acknowledgements

This research was conducted under European Union Sixth Framework Programme Contract FP6-034691 “Net-WMS”.

References

1. A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.
2. Y. Caseau and F. Laburthe. Cumulative scheduling with task intervals. In *Joint International Conference and Symposium on Logic Programming (JICSLP'96)*. MIT Press, 1996.
3. L. Mercier and P. Van Hentenryck. Edge-finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1), 2008.
4. A. Lahrichi. Scheduling: the notions of hump, compulsory parts and their use in cumulative problems. *C.R. Acad. Sci., Paris*, 294:209–211, February 1982.
5. F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers and Operation Research*, 35(3):944–959, 2008.
6. M. Biró. Object-oriented interaction in resource constrained scheduling. *Information Processing Letters*, 36(2):65–67, 1990.
7. F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183(3):1196–1211, 2007.
8. N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. Exhaustive approaches to 2d rectangular perfect packings. *Information Processing Letters*, 90(1):7–14, 2004.
9. P. Van Hentenryck. Scheduling and packing in the constraint language cc(FD). In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.
10. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.
11. N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, and C. Truchet. A generic geometrical constraint kernel in space and time for handling polymorphic k-dimensional objects. In C. Bessière, editor, *Proc. CP'2007*, volume 4741 of *LNCS*, pages 180–194. Springer-Verlag, 2007.
12. Michael R. Garey and David S. Johnson. *Computers and Intractibility. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
13. M. A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118(1–4):73–84, 2003.
14. H. Simonis and B. O'Sullivan. Search strategies for rectangle packing. In P.J. Stuckey, editor, *Proc. CP'2008*, volume 5202 of *LNCS*, pages 52–66. Springer-Verlag, 2008.
15. C. J. Bouwkamp and A. J. W. Duijvestijn. Catalogue of simple perfect squared squares of orders 21 through 25. Technical Report EUT Report 92-WSK-03, Eindhoven University of Technology, The Netherlands, November 1992.