A Generic Geometrical Constraint Kernel in Space and Time for Handling Polymorphic k-Dimensional Objects

N. Beldiceanu[1], M. Carlsson[2], E. Poder[1], R. Sadek[1], and C. Truchet[3]

[1] École des Mines de Nantes, LINA FRE CNRS 2729, FR-44307 Nantes, France
{Nicolas.Beldiceanu,Emmanuel.Poder,Rida.Sadek}@emn.fr
[2] SICS, P.O. Box 1263, SE-164 29 Kista, Sweden
Mats.Carlsson@sics.se
[3] Université de Nantes, LINA FRE CNRS 2729, FR-44322 Nantes, France
Charlotte.Truchet@univ-nantes.fr

SICS Technical Report T2007:08
ISSN: 1100-3154
ISRN: SICS-T–2007/08-SE

Abstract: This report introduces a geometrical constraint kernel for handling the location in space and time of polymorphic k-dimensional objects subject to various geometrical and time constraints. The constraint kernel is generic in the sense that one of its parameters is a set of constraints on subsets of the objects. These constraints are handled globally by the kernel. We first illustrate how to model several placement problems with the constraint kernel. We then explain how new constraints can be introduced and plugged into the kernel. Based on these interfaces, we develop a generic k-dimensional lexicographic sweep algorithm for filtering the attributes of an object (i.e., its shape and the coordinates of its origin as well as its start, duration and end in time) according to all constraints where the object occurs. Experiments involving up to hundreds of thousands of objects and 1 million integer variables are provided in 2, 3 and 4 dimensions, both for simple shapes (i.e., rectangles, parallelepipeds) and for more complex shapes.

Keywords: global constraint, non-overlapping, sweep.

1 Introduction and Presentation of the Kernel

This report introduces a constraint kernel $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{C})$ for handling in a generic way a variety of geometrical constraints $\mathcal{C}$ in space and time between polymorphic k-dimensional objects $\mathcal{O}$ ($k \in \mathbb{N}^+$), where each object takes a shape among a set of shapes described by $\mathcal{S}$ during a given time interval and at a given position in space. This line of research can be seen as a continuation and generalisation of previous work on non-overlapping parallelepipeds [1–4].

Each shape is defined as a finite set of shifted boxes, where each shifted box is described by a box in a k-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* $s = \mathrm{sbox}(sid, t[], l[]) \in \mathcal{S}$ is an entity defined by its shape id $s.sid$, shift offset $s.t[d]$, $0 \leq d < k$, and sizes $s.l[d]$ ($s.l[d] > 0, 0 \leq d < k$). All attributes of a shifted box are integer values. Then, a *shape* is defined as the union of shifted boxes sharing the same shape id. Each *object* $o = \mathrm{object}(id, sid, x[], start, duration, end) \in \mathcal{O}$ is an entity defined by its unique object id $o.id$, shape id $o.sid$, origin $o.x[d]$, $0 \leq d < k$, start in time $o.start$, duration in time $o.duration$ ($o.duration \geq 0$) and end in time $o.end$.[1] All these attributes correspond to domain variables.[2] Typical constraints from the list of constraints $\mathcal{C}$ can express, for instance, the fact that a given subset of objects from $\mathcal{O}$ do not pairwise overlap or that they are all included within a given bounding box. Constraints always have two first arguments $\mathcal{A}_i$ and $\mathcal{O}_i$ (followed by possibly some additional arguments) which resp. specify:

- A list of distinct dimensions (integers in $[0, k-1]$) that the constraint considers.
- A list of identifiers of the objects to which the constraint applies.

Example 1. Assume we have a 3D placement problem (i.e., $k = 3$) involving a set of parallelepipeds $\mathcal{P}$ and one subset $\mathcal{P}'$ of $\mathcal{P}$, where we want to express the fact that (1) no parallelepipeds of $\mathcal{P}$ should overlap, and (2) no parallelepipeds of $\mathcal{P}'$ should be piled. Constraints (1) and (2) resp. correspond to *non-overlapping*$([0,1,2], \mathcal{P})$ and to *non-overlapping*$([0,1], \mathcal{P}')$. Within the first *non-overlapping* constraint, the argument $[0,1,2]$ expresses the fact that we consider a non-overlapping constraint according to dimensions 0, 1 and 2 (i.e., given any pair of parallelepipeds p' and p'' of $\mathcal{P}$ there should exist at least one dimension d ($d \in \{0,1,2\}$) where the projections of p' and p'' on d do not overlap). Similarly, the argument $[0,1]$ of the second non-overlapping constraint expresses the fact that, given any pair of parallelepipeds p' and p'' of $\mathcal{P}'$, there should exist at least one dimension d ($d \in \{0,1\}$) where p' and p'' do not overlap).

$geost(k, \mathcal{O}, \mathcal{S}, \mathcal{C})$ is defined in the following way: given a constraint $ctr_i(\mathcal{A}_i, \mathcal{O}_i)$ from the list of constraints $\mathcal{C}$ between a subset of objects $\mathcal{O}_i \subseteq \mathcal{O}$ according to the attributes $\mathcal{A}_i$, let $\mathcal{MC}_i$ denote the sets of cliques stemming from the objects of $\mathcal{O}_i$ that all overlap in time.[3] The constraints of $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{C})$ hold if and only if $\forall ctr_i \in \mathcal{C}$, $\forall \mathcal{O}_{\mathcal{MC}_i} \in \mathcal{MC}_i : ctr_i(\mathcal{A}_i, \mathcal{O}_{\mathcal{MC}_i})$ holds.

[1] The time dimension is treated specially since the $duration$ attribute may not be fixed, which is not the case for the sizes of a shifted box. Also, the geometrical constraints only apply on objects that intersect in time.

[2] A *domain variable* v is a variable ranging over a finite set of integers denoted by $\mathrm{dom}(v)$; let $\underline{v}$ and $\overline{v}$ resp. denote the minimum and maximum possible values for v.

[3] In fact, these cliques (of an interval graph) are only used for defining the declarative semantics of $geost$'s constraints.

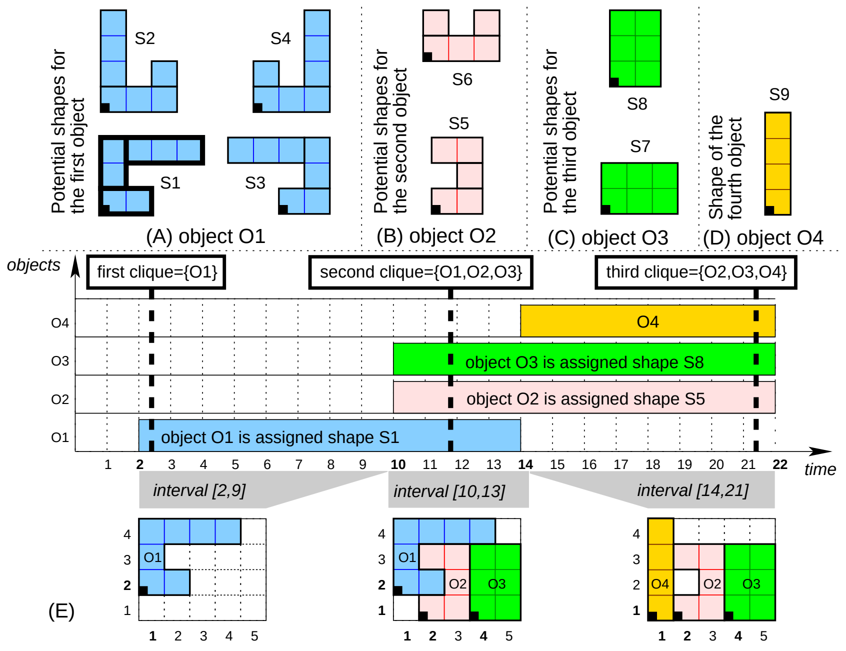


Fig. 1. Example with 4 objects, 9 shapes, one *non-overlapping* and one $included$ constraints

Example 2. Fig. 1 presents a typical example of a dynamic 2D placement problem where one has to place four objects, in time and within a given box, so that objects that overlap in time do not overlap. Parts (A), (B), (C) and (D) resp. represent the potential shapes associated with the four objects to place, where the origin of each object is represented by a black square ■. Part (E) shows the position of the four objects of the example as the time varies, where the first, second, third and fourth objects were resp. assigned shapes S_1, S_5, S_8 and S_9:

– During the first time interval $[2, 9]$ we have only object O_1 at position $(1, 2)$.
– Then, at instant 10 objects O_2 and O_3 both appear. Their origins are resp. placed at positions $(2, 1)$ and $(4, 1)$.
– At instant 14 object O_1 disappears and is replaced by object O_4. The origin of O_4 is fixed at position $(1, 1)$. Finally, at instant 22 all three objects O_2, O_3 and O_4 disappear.

The corresponding arguments are:

```
geost(2,
      [object(1,1,[1,2], 2,12,14), object(2,5,[2,1],10,12,22),
       object(3,8,[4,1],10,12,22), object(4,9,[1,1],14, 8,22)],
      [sbox(1,[0,0],[2,1]), sbox(1,[0,1],[1,2]), sbox(1,[1,2],[3,1]),
       sbox(2,[0,0],[3,1]), sbox(2,[0,1],[1,3]), sbox(2,[2,1],[1,1]),
       sbox(3,[0,0],[2,1]), sbox(3,[1,1],[1,2]), sbox(3,[2,2],[3,1]),
       sbox(4,[0,0],[3,1]), sbox(4,[0,1],[1,1]), sbox(4,[2,1],[1,3]),
       sbox(5,[0,0],[2,1]), sbox(5,[1,1],[1,1]), sbox(5,[0,2],[2,1]),
       sbox(6,[0,0],[3,1]), sbox(6,[0,1],[1,1]), sbox(6,[2,1],[1,1]),
       sbox(7,[0,0],[3,2]), sbox(8,[0,0],[2,3]), sbox(9,[0,0],[1,4])],
      [non-overlapping([0,1],[1,2,3,4]),included([0,1],[1,2,3,4],[1,1],[5,4])])
```

Its first argument 2 is the number of dimensions of the placement space we consider. Its second and third arguments resp. describe the four objects and the shifted boxes of the nine shapes we have. For instance, the 3 boxes of shape S_1 (depicted by 3 thick rectangles in Part (A) of Fig. 1) respectively correspond to the 3 boxes declared at line 04 of the example. Finally, its last argument gives the list of geometrical constraints imposed by $geost$: the first constraint expresses a non-overlapping constraint between the four objects, while the second constraint imposes the four objects to be located within the box containing all points (x, y) such that $1 \leq x \leq 1+5-1$ and $1 \leq y \leq 1+4-1$. The constraints of $geost$ hold since the four objects do not simultaneously overlap in time and in space and since they are completely included within the previous box (i.e., see Part (E) of Fig. 1).

Within the scope of $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{C})$, this report presents a filtering algorithm that prunes the domain of each attribute of every object $o = \text{object}(id, sid, x[], start, duration, end) \in \mathcal{O}$. All values found infeasible are deleted from the shape attribute sid; for the other attributes (i.e., the origin $x[]$, the start, the duration and the end), the minimum and maximum are adjusted. The approach presented in this report offers a number of advantages:

- The main theoretical advantages are fourfold:
  - First, the geometrical kernel makes it possible to integrate new geometrical constraints as new applications and/or requirements show up. This is achieved by providing for each geometrical constraint an API without knowing any details about the geometrical kernel. This contrasts with traditional approaches where one has to come up with a rather involved filtering algorithm for each global constraint.
  - Second, while pruning the attribute of an object, the geometrical kernel takes direct advantage of all geometrical constraints involving that object in order to perform more deduction. This is a fundamental progress over the traditional approach where constraints only co-operate through the domains of their shared variables.
  - Even when we have three or four dimensions, the approach scales well since it does not rely on building complex multi-dimensional data structure (e.g., like quadtrees or octrees). It only stores a number of points in the order of $O(m \cdot k)$ where m is the total number of objects and k is the number of dimensions.
  - Even if complex objects could be decomposed into boxes for which one links the coordinates by external equality constraints this weakens a lot the deduction process as illustrated by the following example of Figure 2: if the shape s (see Part (A)) is decomposed into two rectangles $r4$ and $r5$ (see Part (C)) and if the constraints linking the coordinates of the origins of $r4$ and $r5$ are not integrated within the sweep process, infeasibility cannot be directly derived (see Part (M)). In contrast our approach allows to detect infeasibility directly by reasoning only on the coordinates of the origin of s.
- The main practical advantages are as follows:
  - Having k dimensions allows to come up with a single constraint that can be used for handling general non-overlapping constraints. This was originally motivated by a warehouse management problem where both two-dimensional and three-dimensional sub-problems had to be solved. In the context of three-dimensional packing problems having an extra dimension also makes sense for modelling the fact that we want to assign objects to a truck (in this context we

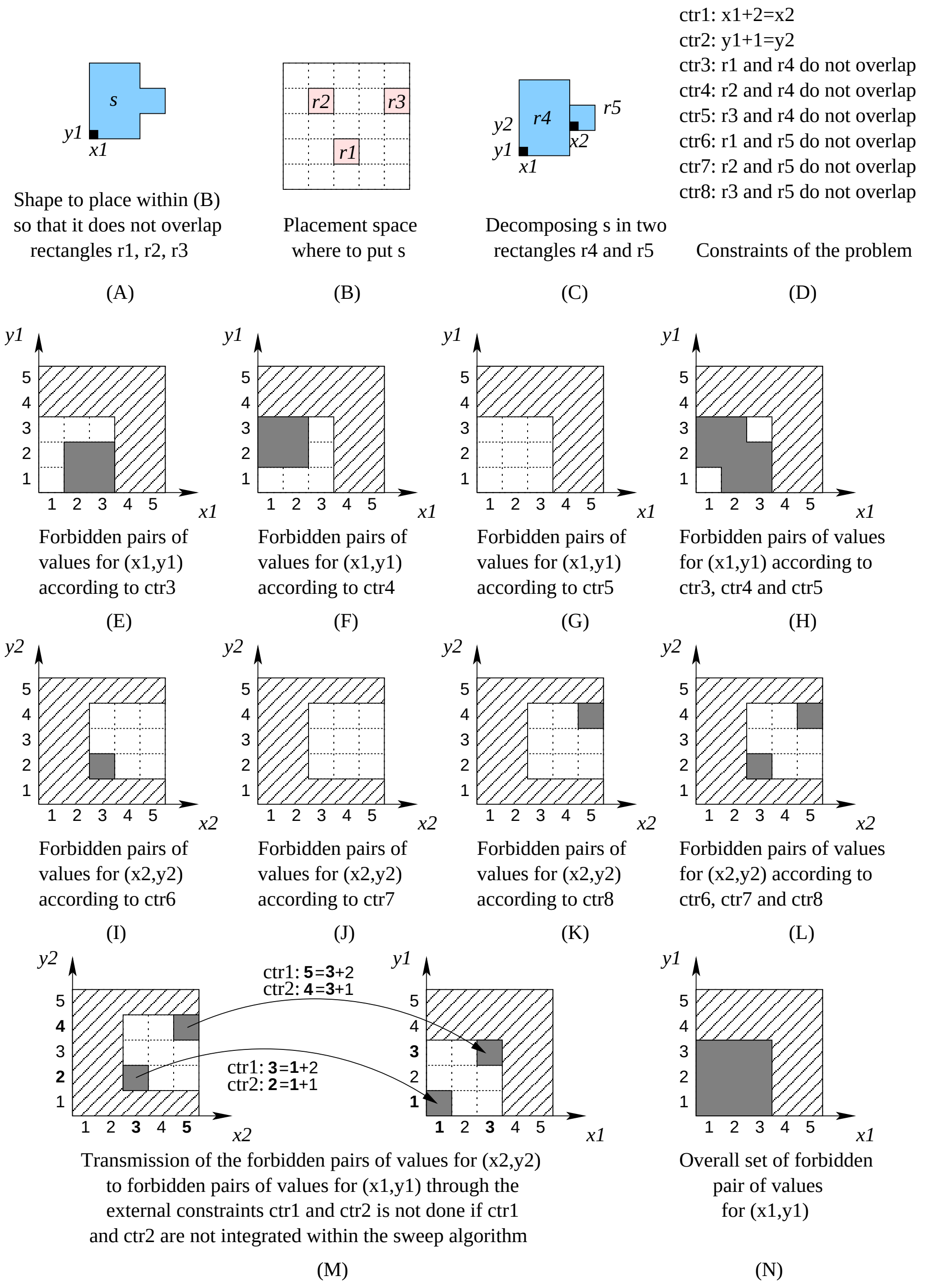

s
y1
x1
Shape to place within (B) so that it does not overlap rectangles r1, r2, r3
(A)
r2
r3
r1
Placement space where to put s
(B)
r4
r5
y2
y1
x1
x2
Decomposing s in two rectangles r4 and r5
(C)
ctr1: x1+2=x2
ctr2: y1+1=y2
ctr3: r1 and r4 do not overlap
ctr4: r2 and r4 do not overlap
ctr5: r3 and r4 do not overlap
ctr6: r1 and r5 do not overlap
ctr7: r2 and r5 do not overlap
ctr8: r3 and r5 do not overlap
Constraints of the problem
(D)
Forbidden pairs of values for (x1,y1) according to ctr3
(E)
Forbidden pairs of values for (x1,y1) according to ctr4
(F)
Forbidden pairs of values for (x1,y1) according to ctr5
(G)
Forbidden pairs of values for (x1,y1) according to ctr3, ctr4 and ctr5
(H)
Forbidden pairs of values for (x2,y2) according to ctr6
(I)
Forbidden pairs of values for (x2,y2) according to ctr7
(J)
Forbidden pairs of values for (x2,y2) according to ctr8
(K)
Forbidden pairs of values for (x2,y2) according to ctr6, ctr7 and ctr8
(L)
ctr1: 5=3+2
ctr2: 4=3+1
ctr1: 3=1+2
ctr2: 2=1+1
Transmission of the forbidden pairs of values for (x2,y2) to forbidden pairs of values for (x1,y1) through the external constraints ctr1 and ctr2 is not done if ctr1 and ctr2 are not integrated within the sweep algorithm
(M)
Overall set of forbidden pair of values for (x1,y1)
(N)

Fig. 2. Reasoning for detecting the infeasibility of the placement problem (dashed areas correspond to initially forbidden pairs of values, while grey areas represent forbidden pairs of values related to some non-overlapping constraints)

speak about an *assignment dimension* – see Part (H) of Figure 3) or the fact that we do not want to place all the objects since there may simply be not enough room (in this context we speak about a *relaxation dimension*).

- Factoring out the description of the shapes from the description of an object makes sense in a lot of practical problems where a number of instances of the same shape have to be considered (this is illustrated by Part (I) of Figure 3 where we have five objects but only three shapes: in fact the first, third and fifth objects correspond to the first shape). This again occurs in the warehouse management problem that originally motivated the constraint, where a major car manufacturer has to pack within the same container the parts associated with 24 instances of the same car model. By doing so we can decrease the memory requirement (i.e., each complex shape is represented only once).
- Having a set of potential shapes for an object offers an extra modelling power for representing directly the fact that objects may rotate, or for dealing with tasks for which the duration depends on the machine where the task is actually assigned.
- Having a temporal dimension allows to tackle dynamic placement problems where objects are moving in time. Consider for instance a pick-up delivery problem where objects are loaded or unloaded from a truck while visiting different locations. In this context the non-overlapping constraint applies only for those objects which overlap in time. This is illustrated by Part (I) of Figure 3.

The report is organised as follows. Section 2 provides an overview of placement problems that can be modelled with the constraints currently available in $geost$. Section 3 presents the overall architecture of the geometrical kernel. It explains how to define geometrical constraints in terms of a programming interface by the geometrical kernel. Section 4 focusses on the main contribution of this report: a multi-dimensional lexicographic sweep algorithm used for filtering the attributes of an object of $geost$. Section 5 evaluates the scalability of the $geost$ kernel as well as its ability to deal with a variety of specific placement problems. Before we conclude, Section 6 compares $geost$ with related work and suggests future directions. Finally an annex provides the data sets and test programs used in Section 5.

2 Modelling Problems with $geost$

As illustrated by Fig. 3 in the context of *non-overlapping*, $geost$ allows to model directly a large number of placement problems:

– Case (A) corresponds to a non-overlapping constraint among three segments.
– The second and third cases (B,C) correspond to a non-overlapping constraint between rectangles where (B) is a special case where the sizes of all rectangles in the second dimension are equal to 1; this can be interpreted as a *machine assignment problem*.
– Case (D) corresponds to a non-overlapping constraint between rectangles where each rectangle can have two orientations. This is achieved by associating with each rectangle two shapes of respective sizes $l \times h$ and $h \times l$. Since their orientation is not

initially fixed, the *included* constraint enforces the three rectangles to be included within the bounding box defined by the origin's coordinates $1, 1$ and sizes $8, 3$.

- Case (E) corresponds to a non-overlapping constraint between more complex objects where each object is described by a given set of rectangles.
- Case (F) describes a placement problem where one has to first assign each rectangle to a strip so that all rectangles that are assigned to the same strip do not overlap.
- Case (G) corresponds to a non-overlapping constraint between parallelepipeds.
- Case (H) can be interpreted as a non-overlapping constraint between parallelepipeds that are assigned to the same container. The first dimension corresponds to the identifier of the container, while the next three dimensions are associated with the position of a parallelepiped inside a container.
- Case (I) describes a rectangle placement problem over three consecutive time-slots: rectangles assigned to the same time-slot should not overlap in time. We initially start with the three rectangles 1, 2 and 3. Rectangle 3 is no longer present at instant 2 (the triangle ▼ within rectangle 3 at time 1 indicates that rectangle 3 will disappear at the next time-point), while rectangle 4 appears at instant 2 (the triangle ▲ within rectangle 4 at time 2 denotes the fact that the rectangle 4 appears at instant 2). Finally, rectangle 2 disappears at instant 3 and is replaced by rectangle 5.

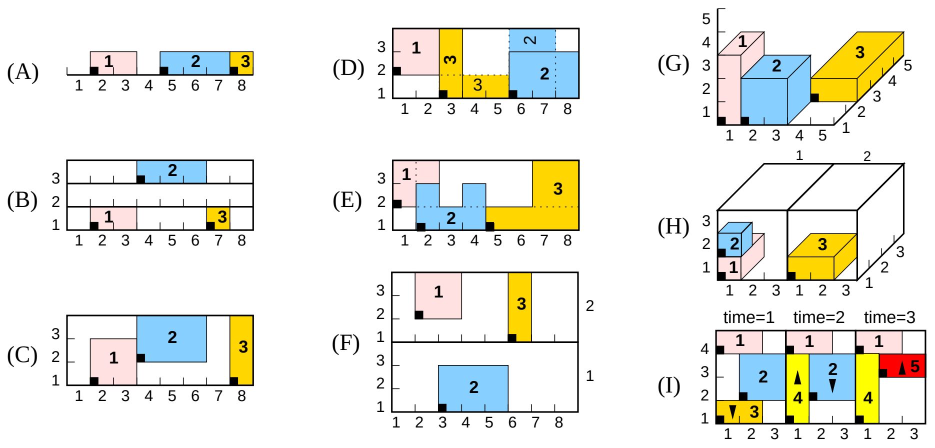


Fig. 3. Nine typical examples of use of *geost*

3 Standard Representation of Geometrical Constraints

The key idea for handling multiple geometrical constraints in a common kernel is the following. For each type of geometrical constraint found in $\mathcal{C}$ (also called *external* constraints), one has to provide a service that computes necessary conditions (also called *internal* constraints) for a given object and shape. Given an external geometrical constraint $ectr_i(\mathcal{A}_i, \mathcal{O}_i)$ ($\mathcal{A}_i \subseteq \{0, 1, \ldots, k-1\}, \mathcal{O}_i \subseteq \mathcal{O}$), one of its object $o \in \mathcal{O}_i$

and one potential shape s of o, such a necessary condition generated by $ectr_i$, o and s is a unary[4] constraint $ictr(o.x)$ such that: $o.sid = s \wedge ectr_i(\mathcal{A}_i, \mathcal{O}_i) \Rightarrow ictr(o.x)$. Now, the key to being able to globally treat such necessary conditions in the kernel is to give them a uniform representation. We have chosen the following one:

– A constraint $\mathrm{outbox}(t, l)$ on $o.x$ holds iff $o.x$ is located outside the shifted box defined by its origins point $t[d]$, $0 \leq d < k$, and sizes $l[d]$, $0 \leq d < k$ (i.e., $\exists d \in [0, k-1] \mid o.x[d] < t[d] \vee o.x[d] > t[d] + l[d] - 1$).

Thus, an outbox corresponds to a box-shaped set of points that are infeasible for $o.x$. The purpose of the introduction of outboxes is to have a common representation for the kernel, suitable for the k-dimensional lexicographic sweep algorithm presented in the next section, which considers all the outboxes, for a selected object and shape, in one run.

Consequently, for each type of external geometrical constraint, found in $\mathcal{C}$ a service $\mathrm{GenOutboxes}(ectr_i, o, s) : (ictrs)$, responsible for generating outboxes, must be provided. This service is assumed to generate outboxes that intersect the domains of the origin coordinates of o. Also, if all attributes mentioned by $ectr_i$ belonging to objects other than o are fixed, those outboxes are assumed to be necessary and sufficient conditions, lest the kernel accept false solutions.

Example of External Geometrical Constraints. We now illustrate some external geometrical constraints that are currently available within the constraint kernel. As we saw in the introduction, an external constraint always has at least two arguments that resp. correspond to a list of distinct dimensions and to a list of object identifiers to which the constraint apply.

The $included$ and *non-overlapping* external constraints. The $included(\mathcal{A}_i, \mathcal{O}_i, t, l)$ and the *non-overlapping*$(\mathcal{A}_i, \mathcal{O}_i)$ external constraints take as input a list of distinct dimensions $\mathcal{A}_i$ in $\{0, 1, \ldots, k-1\}$ and a list $\mathcal{O}_i$ of distinct object identifiers of $geost$. In addition, the $included$ constraint considers a shifted box defined by its origin point $t[d]$, $0 \leq d < k$, and size $l[d]$, $0 \leq d < k$.

The $included$ constraint enforces for each object o (with $o.id \in \mathcal{O}_i$) and for any corresponding shifted box s (with $o.sid = s.sid$) the condition $\forall d \in \mathcal{A}_i \mid t[d] \leq o.x[d] + s.t[d] \wedge o.x[d] + s.t[d] + s.l[d] - 1 \leq t[d] + l[d] - 1$ (i.e., s is included within the shifted box attribute defined by the parameters t and l of the $included$ constraint). Depending on which shape of an object we actually consider, the $included$ constraint can be translated to $2k$ outbox constraints.

The *non-overlapping* constraint enforces the following condition: given two distinct objects o and o' (with $o.id, o'.id \in \mathcal{O}_i$) that overlap in time, no shifted box s (with $o.sid = s.sid$) should overlap any shifted box s' (with $o'.sid = s'.sid$); i.e. it should hold that $\exists d \in \mathcal{A}_i \mid o.x[d] + s.t[d] + s.l[d] \leq o'.x[d] + s'.t[d] \vee o'.x[d] + s'.t[d] + s'.l[d] \leq o.x[d] + s.t[d]$ (i.e., there exists a dimension where they do not intersect). While focussing on an object o we can easily generate an outbox constraint for each object o' that should not overlap o by reusing the results of [2].

[4] Unary, since it involves the k coordinates of a *single* object.

4 The Geometrical Kernel: a Generic k-dimensional Lexicographic Sweep Algorithm

In this section, we first present the sweep algorithm used for filtering the coordinates of the origin of an object o of $geost$ when each object has one single shape. We initially assume that time is treated exactly like the space dimensions, i.e. that the $o.x$ array is extended by one element. Toward the end of this section, we explain in detail how to treat the time attributes of an object. We also assume for now that the shape attribute is fixed, and explain later how to handle multiple potential shapes for an object (i.e., polymorphism). We now introduce some notation used throughout this section.

Notation. Assume v and w are vectors of scalars of k components. Then $v \leftarrow w$ denotes the element-wise assignment of w to v, $w + d$ (resp. $w - d$) denotes the element-wise addition of d (resp. $-d$) to w. Given a scalar d, $0 \leq d \leq k-1$, $\mathrm{rot}(v, d, k)$ denotes the vector $(v[d], v[(d+1) \mod k], \ldots, v[(d-1) \mod k])$. That is, in the rotated vector, $v[d]$ is the most significant element, which is what we need when running the sweep algorithm on dimension d.

The Sweep Algorithm. This algorithm first considers all outboxes $\mathcal{IC}_o$ derived from $\mathcal{C}$ where object o actually appears, and then performs a recursive traversal of the placement space for each coordinate and direction (i.e., $\min$ or $\max$). Without loss of generality, assume we want to adjust the minimum value of the d^{th} coordinate $o.x[d]$, $0 \leq d < k$, of the origin of o. The algorithm starts its recursive traversal of the placement space at point $c = \mathrm{rot}(\underline{o.x}, d, k)$ and could in principle explore all points of the domains of $o.x$, one by one, in increasing lexicographic order, until a point is found that is not inside any outbox, in which case $c[0]$ is the computed new minimum value. To make the search efficient, instead of moving each time to the successor point, we arrange the search so that it skips points that are known to be inside some outbox.[5]

Thus, we compute the lexicographically smallest point c' such that:

1. c' is lexicographically greater than or equal to c,
2. every element of c' is in the domain of the corresponding element of $o.x$,
3. c' is not inside any outbox of $\mathcal{IC}_o$.

If no such c' exists, the constraint fails. Otherwise, the minimum value of $o.x[d]$ is adjusted to $c'[0]$. As we saw, the sweep algorithm moves in increasing lexicographic order a point c from its lexicographically smallest potential feasible position to its lexicographically largest potential feasible position through all potential points. The algorithm uses the following data structures:

– The current position c of the sweep.
– A vector $n[0..k-1]$ that records knowledge about already encountered sets of infeasible points while moving c from its first potential feasible position. The vector n is always element-wise greater than c and maintained as follows. Let $\inf, \sup$ denote the vectors $\inf = \mathrm{rot}(\underline{o.x}, d, k)$ and $\sup = \mathrm{rot}(\overline{o.x} + 1, d, k)$:

[5] Potential holes in the domains are reflected in outboxes.

- Initially, $n = \sup$.
- Whenever an outbox f containing c is found, n is updated by taking the element-wise minimal value of n and the upper boundary of $\mathrm{rot}(f, d, k)$, indicating the fact that new candidate points can be found beyond that value.
- Whenever we skip to the next candidate point, we reset the elements of n that were used to the corresponding values of $\sup$.

The following invariant holds for the vector n, and is used when advancing c to the next candidate point. Let i be the smallest j such that $n[j+1] = \sup[j+1] \wedge \cdots \wedge n[k-1] = \sup[k-1]$ and suppose c is known to be in some outbox. Then, the next point, lexicographically greater than c and not yet known to be in any outbox, is $(c[0], \ldots, c[i-1], n[i], \inf[i+1], \ldots, \inf[k-1])$.

Algorithm 1 implement this idea. The algorithm prunes the bounds of each coordinate of every object wrt. its relevant outboxes, iterating to fix-point.

Efficiency. The main inefficiency in this sweep algorithm lies in searching the set of outboxes (line 4 of $\mathrm{PruneMin}$). In order to make this search more efficient, we can make the sweep algorithm more sophisticated by the following modifications to $\mathrm{PruneMin}$:

– We extend the state of the algorithm by an *event point series*, ordered in lexicographically increasing order. These events correspond to the lexicographically smallest (insert events) and largest (delete events) relevant infeasible point associated with each outbox $ictr_o \in \mathcal{IC}_o$. They are sorted in lexicographically increasing order, and we maintain a pointer into the series in sync with point c.
– We maintain the set of *active outboxes*, corresponding to all outboxes $ictr_o \in \mathcal{IC}_o$ such that c is between its lexicographically smallest and largest infeasible points. This set is initially empty.
– When c is initialized in line 2 as well as when c is incremented in lines 7-17, the relevant events up to point c from the event point series are processed, and the corresponding outboxes are added to or deleted from the set of active outboxes.
– In line 4, only the active outboxes are considered.

Example 3. Fig. 4 illustrates the k-dimensional lexicographic sweep algorithm in the context of $k = 2$. Parts (A) and (B) provide the variables of the problem (i.e., the abscissa and ordinate of each rectangle r_1, r_2, r_3, r_4 and r_5) as well as the non-overlapping constraint between the five previous rectangles. On Part (D) we have represented the extreme possible feasible positions of each rectangle (i.e., rectangles r_1 to r_4): for instance the leftmost lower corner of rectangle r_1 can only be fixed at positions $(1, 2)$, $(1, 3)$, $(1, 4)$, $(2, 2)$, $(2, 3)$, $(2, 4)$, $(3, 2)$, $(3, 3)$, $(3, 4)$, $(4, 2)$, $(4, 3)$ and $(4, 4)$. Parts (C) to (L) of Fig. 4 detail the different steps of the algorithm for adjusting the minimum value of the abscissa of rectangle r_5. Part (C) provides the outboxes associated with the fact that we want to prune the coordinates of r_5: constraints ctr_1, ctr_2, ctr_3 and ctr_4 resp. correspond to the fact that rectangle r_5 should not overlap rectangles r_1, r_2, r_3 and r_4, while constraint ctr_5 represents the fact that the ordinate of r_5 should be different from 7. Part (D) represents the initialisation phase of the algorithm where we have all five outboxes with their respective lexicographically smallest infeasible point (i.e., $(1, 1)$ for ctr_1, $(1, 3)$ for ctr_2, $(1, 7)$ for ctr_5, $(1, 8)$ for ctr_3 and $(3, 1)$ for ctr_4). Part (E) represents the first step of the sweep algorithm where we start the traversal of the placement space at point $c = (1, 1)$. We first

VARIABLES
x1 in 1..4, y1 in 2..4
x2 in 4..4, y2 in 6..6
x3 in 2..4, y3 in 8..9
x4 in 7..7, y4 in 1..1
x5 in 1..8, y5 in 1..8, y5<>7

(A)

EXTERNAL CONSTRAINT (non−overlapping)

```
geost( [object(1,1,[x1,y1],0,1,1),object(2,2,[x2,y2],0,1,1),
        object(3,3,[x3,y3],0,1,1),object(4,4,[x4,y4]0,1,1),
        object(5,5,[x5,y5],0,1,1)],
       [shape(1,[0,2],[0,1]),shape(2,[0,3],[0,1]),shape(3,[0,1],[0,1]),
        shape(4,[0,1],[0,3]),shape(5,[0,5],[0,4])],
       [non−overlapping([0,1],[1,2,3,4,5])] )
```

(B)

INTERNAL CONSTRAINTS GENERATED FOR FILTERING THE ORIGIN OF THE FIFTH OBJECT, i.e. (x5,y5) (ICTRS)
ctr1: outbox([1,1],[2,2]) ctr3: outbox([1,8],[2,1])
ctr2: outbox([1,3],[6,4]) ctr4: outbox([3,1],[5,3])
ctr5: outbox([1,7],[8,1])

(C)

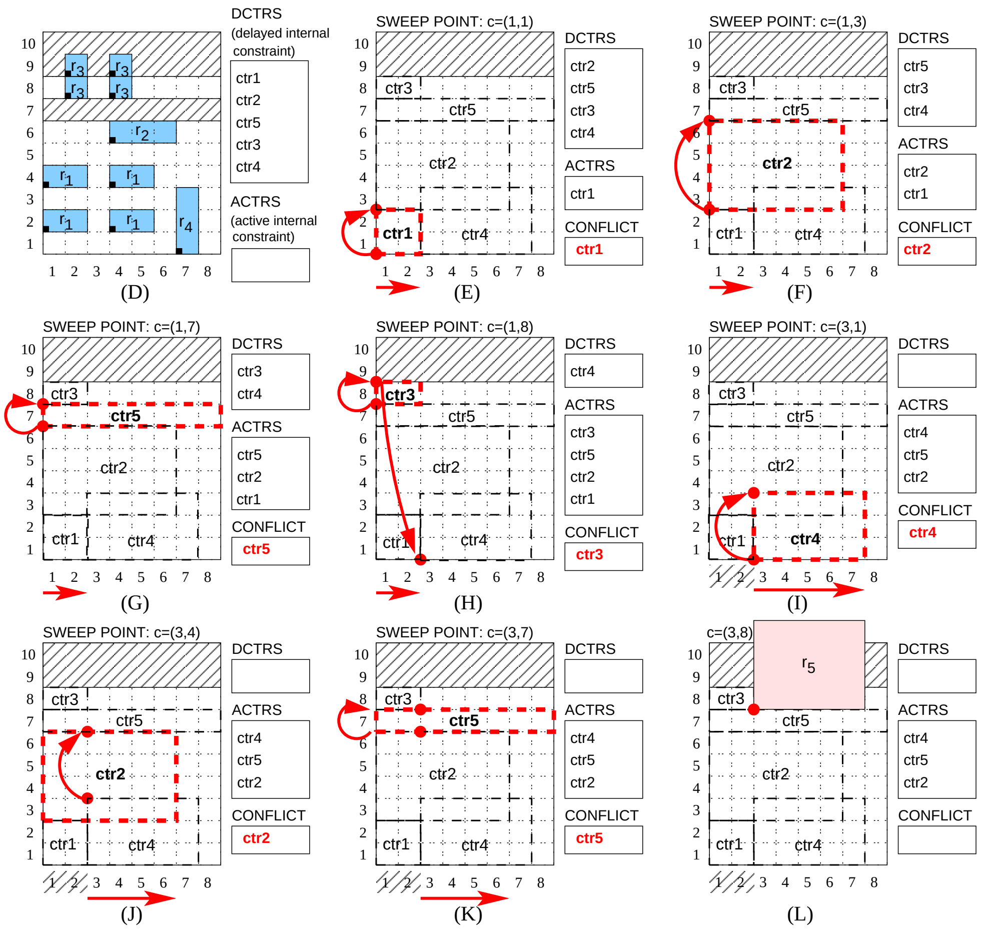


Fig. 4. Illustration of the lexicographic sweep algorithm for adjusting the minimum value of the abscissa of rectangle r_5

```
PROCEDURE FilterCtrs($k, \mathcal{O}, \mathcal{S}, \mathcal{C}$) : bool
 1: $nonfix \leftarrow \mathbf{true}$                                   // fixpoint not yet reached
 2: while $nonfix$ do
 3:   $nonfix \leftarrow \mathbf{false}$                                // assumes no filtering will be done
 4:   for all $o \in O$ do
 5:     $I \leftarrow \bigcup_{e \in \mathcal{C}} \mathrm{GenOutboxes}(e, o, o.\mathrm{sid})$   // build the set of outboxes on $o$
 6:     $I \leftarrow I \cup \bigcup_{0 \leq d < k}$ possible outboxes corresponding to holes in $o.x[d]$
 7:     for $d \leftarrow 0$ to $k-1$ do
 8:       if $\neg\mathrm{PruneMin}(o,d,k,I) \vee \neg\mathrm{PruneMax}(o,d,k,I)$ then
 9:         return false                                                // no feasible origin
10:       else if $o.x$ was pruned then
11:         $nonfix \leftarrow \mathbf{true}$                           // fixpoint not yet reached
12:       end if
13:     end for
14:   end for
15: end while
16: return $\mathbf{true}$                                               // feasible origin
PROCEDURE PruneMin($o, d, k, I$) : bool
 1: $b \leftarrow \mathbf{true}$                                        // $b = \mathbf{true}$ while we have not failed
 2: $c \leftarrow \underline{o.x}$                                      // initial position of the point
 3: $n \leftarrow \overline{o.x} + 1$                                   // upper limits+1 in the different dimensions
 4: while $b \wedge \exists f \in I \mid c \in f$ do
 5:   $n \leftarrow \min(n, f.t + f.l)$                 // update vector $n$ according to an outbox $f$ containing $c$
 6:   $b \leftarrow \mathbf{false}$                                     // no new point to jump to yet
 7:   for $j \leftarrow k-1$ downto $0$ do
 8:     $j' \leftarrow (j+d) \mod k$                                    // rotation wrt. $d, k$
 9:     $c[j'] \leftarrow n[j']$                                        // use vector $n$ to jump
10:     $n[j'] \leftarrow \overline{o.x}[j'] + 1$                       // reset component of $n$ to maximum value
11:     if $c[j'] \leq \overline{o.x}[j']$ then
12:       $b \leftarrow \mathbf{true}$                                  // jump target found
13:       $j \leftarrow 0$                                              // exit for loop
14:     else
15:       $c[j'] \leftarrow \underline{o.x}[j']$                        // reset component of $c$, for exhausted a dimension
16:     end if
17:   end for
18: end while
19: if $b$ then
20:   $\underline{o.x[d]} \leftarrow \max(\underline{o.x[d]}, c[d])$
21: end if
22: return $b$
```

Algorithm 1: FilterCtrs is the main filtering algorithm associated with $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{C})$, where k, $\mathcal{O}$, $\mathcal{S}$ and $\mathcal{C}$ resp. correspond to the number of dimensions, to the objects, to the shapes and to the external geometrical constraints. PruneMin adjusts the lower bound of the d^{th} coordinate of the origin of object o where I is the set of outboxes associated with object o (since PruneMax is similar to PruneMin it is omitted). The given fixpoint loop is an over-simplification. The implementation maintains a set of objects that need filtering. Whenever an object o is pruned, all non-fixed objects connected to o by an external constraint are added to this set. When the set becomes empty, the fixpoint is reached.

transfer to the list of active outboxes all outboxes for which the first lexicographically smallest infeasible point is lexicographically greater than or equal to the current position of the sweep $c = (1, 1)$ (i.e., constraint $ctr_1 = \mathrm{outbox}([1, 1], [2, 2])$). We then search through the list of active constraints (represented on the figure by a box with the legend ACTRS on top of it) the first constraint for which $c = (1, 1)$ is infeasible. In fact, since ctr_1 is infeasible (represented on the figure by a box with the legend CONFLICT on top of it) we compute the vector $f = (3, 3)$ that tells how to get the next potentially feasible point in the different dimensions. Consequently the sweep moves to the next position $(1, 3)$ (see Part (F)) and the process is repeated until we finally find a feasible point for all outboxes (i.e., point $(3, 8)$ in Part (L)). Note that, when the lexicographically largest infeasible point associated with an active outbox is lexicographically less than the current position of the sweep, we remove that constraint from the list of active outboxes. This is for instance the case in Part (I), where we remove constraint ctr_3 from the list of active outboxes (since its lexicographically largest infeasible point $(2, 8)$ is lexicographically less than the position of the sweep $c = (3, 1)$).

Complexity. Rather than analysing the complexity of the $geost$ kernel for a fixed k, which depends both on the type of each external constraint (i.e., the complexity of a given external constraint for generating all its corresponding outboxes as well as their number), we rather focus on $\mathrm{PruneMin}$ for adjusting the minimum value of the d^{th} coordinate of the origin of an object. Assuming that the maximum number of outboxes is equal to n we give an upper bound on the maximum number of jumps of $\mathrm{PruneMin}$ (i.e., the maximum number of times the sweep is moved).

First note that we always jump to an upper border (+1) of an outbox (i.e., see line 5 of $\mathrm{PruneMin}$) or that we reset some coordinates of the sweep to its minimum value (i.e., see line 15 of $\mathrm{PruneMin}$). Consequently, all coordinates of the sweep are always equal to an upper border (+1) of some outboxes or to a minimum possible value. Since we want to evaluate the maximum number of jumps, let us assume that for every dimension d ($0 \leq d < k$) the upper limits of all the n outboxes are distinct. Having this in mind we can construct a maximum of $(n + 1)^k$ points. Even if we found a systematic construction where this number of jumps is reached, the performance evaluation of Section 5 indicates that we can handle a reasonable number of objects for $k = 2, 3, 4$.

From a memory consumption point of view, the algorithm only records the coordinates of the sweep from one invocation to the next, in order not to restart the search from scratch (i.e., $2k$ points for each object).

Handling Time. Given an object $o \in \mathcal{O}$ of $geost$, the sweep algorithm that we have introduced in the previous section can be easily adapted to handle the start in time $o.start$, duration in time $o.duration$ and end in time $o.end$. Beside maintaining bound consistency for the constraint $o.end = o.start + o.duration$, we add an extra *time* dimension to the geometric coordinates of object o. Roughly, this new time coordinate corresponds to $o.start$ resp. $o.end$ depending on whether we are adjusting the minimum or maximum.

Handling Polymorphism. In order to handle the fact that objects can have several potential shapes we modify the previous algorithm in the following way. For adjusting the minimum value of the coordinate of the origin of an object that has more than one shape we call the sweep algorithm for each potential shape of the object (i.e., for each value of its shape variable). Then we take the smallest minimum value obtained (i.e., we use constructive disjunction) and prune the shape variable of an object if we did not find any feasible point for a given potential shape of that object.

Other Internal Constraints. The standard representation of geometrical constraints given in Section 3 is an over-simplification. For some constraints, e.g. distance constraints, outboxes are not a suitable representation, as the set of forbidden coordinates cannot be covered by a small number of boxes. Therefore, the constraint kernel internally rather handles internal constraints, with an appropriate internal API. In order to have a compact representation which can be used efficiently by the geometrical kernel a set of infeasible points is in fact defined implicitly by providing the following functions:

– $\mathrm{LexInfeasible}(ictr, minlex, d, k, o) : (found, p)$ when $minlex = \mathbf{true}$ (respectively $\mathbf{false}$), returns the smallest (respectively largest) infeasible lexicographical point p associated with the internal geometrical constraint $ictr$ (according to the fact that we prune the d^{th} coordinate of the origin of object o, i.e., the ordering among the different dimensions is $d, (d+1) \mod k, \ldots, (d-1) \mod k$) compatible with the domains of the coordinates of the origin of o. If no such point exists, $found$ is set to $\mathbf{false}$ (otherwise $found$ is set to $\mathbf{true}$).
– $\mathrm{IsFeasible}(ictr, min, d, k, o, c) : (feasible, f)$ sets $feasible$ to $\mathbf{true}$ if point c is feasible according to the internal constraint $ictr$; if this is not the case, sets $feasible$ to $\mathbf{false}$, and computes the forbidden region f according to the fact that we prune the minimum ($min = \mathbf{true}$) or the maximum ($min = \mathbf{false}$) value of the d^{th} coordinate of o: we first maximise the size of f in dimension $(d-1) \mod k$, then maximise the size of f in dimension $(d-2) \mod k$ and so on until we reach the most significant dimension d. Part (A) (respectively Part (B)) of Figure 5 illustrates the computation of the forbidden region f in the context of $k = 2$ and $d = 0$ (respectively $d = 1$).
– $\mathrm{CardInfeasible}(ictr, k, o) : (n)$ returns an estimation of the number n of infeasible points for the origin of object o under the assumption that constraint $ictr$ holds. This information is used as a heuristics for ordering the internal constraints checked by the geometrical kernel.

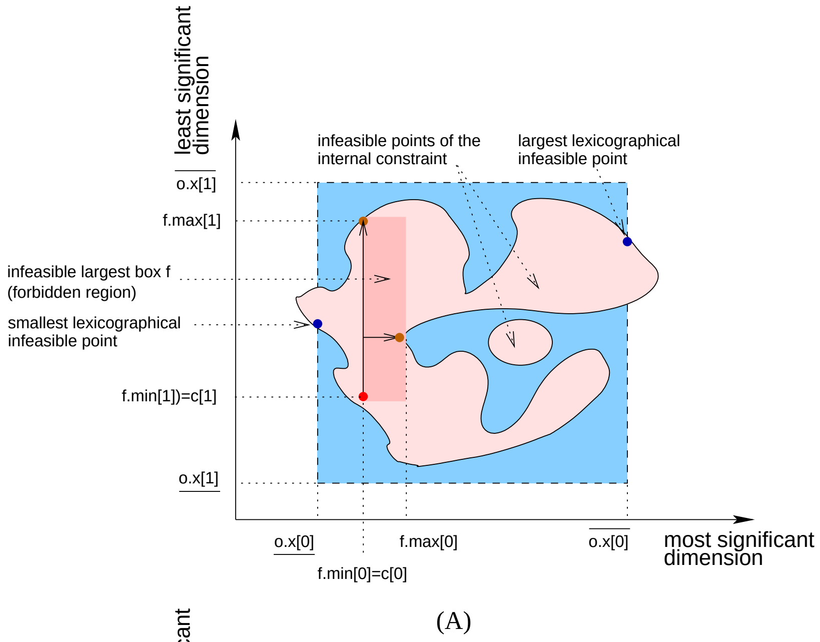
Algorithm 1 can be adapted to this API: (1) The variable I initialised at lines 5 and 6 of $\mathrm{FilterCtrs}$ would be a set of internal constraints instead of outboxes. (2) Line 4 of $\mathrm{PruneMin}$ would use $\mathrm{IsFeasible}$.

5 Performance Evaluation

We evaluate the implementation[6] of the *geost* kernel from three perspectives:

Wanting to measure the speed and the scalability of the sweep algorithm for finding a first solution on loosely constrained placement problems (i.e., 20% spare space), we generated one set of random problem instances of m k-dimensional boxes for

[6] The experiments were run in SICStus Prolog 4 compiled with gcc -02 version 4.0.2 on a 3GHz Pentium IV with 1MB of cache.


least significant dimension
infeasible points of the internal constraint
largest lexicographical infeasible point
o.x[1]
f.max[1]
infeasible largest box f (forbidden region)
smallest lexicographical infeasible point
f.min[1])=c[1]
o.x[1]
o.x[0]
f.max[0]
o.x[0]
most significant dimension
f.min[0]=c[0]
(A)

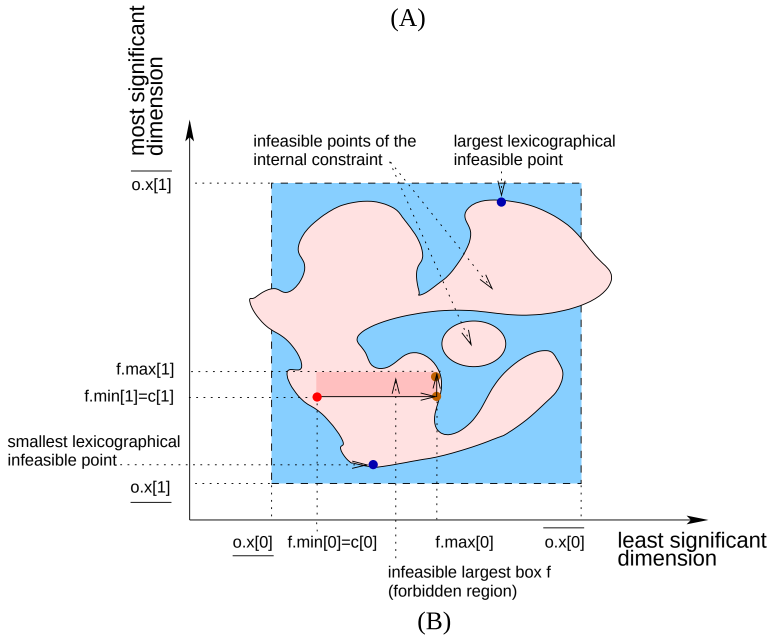

most significant dimension
infeasible points of the internal constraint
largest lexicographical infeasible point
o.x[1]
f.max[1]
f.min[1]=c[1]
smallest lexicographical infeasible point
o.x[1]
o.x[0]
f.min[0]=c[0]
f.max[0]
o.x[0]
least significant dimension
infeasible largest box f (forbidden region)
(B)

Fig. 5. Illustration of an internal geometrical constraint and of how to compute the forbidden region according to the dimension for which we want to prune (i.e., the most significant dimension)

$k \in \{2,3,4\}$ involving $t \in \{1,16,256,1024\}$ distinct types of boxes, and for $m \in \{1024, 2048, \ldots, 262144\}$. The results for $k = 2$ are shown in Fig. 6 (top left) and indicates that the approach is sensible to the number of distinct types of boxes. It can typically pack 1024 2D, 3D and 4D distinct boxes in at most 200 msec. The longest time, 13694 seconds (close to 4 hours), was obtained for packing 262144 4D parallelepipeds (over 1 million domain variables) with a memory consumption of 351MB.

Wanting to get an idea of the performance of the $geost$ kernel on very tight placement problems (i.e., 0% spare space), we considered the *perfect squared squares problem* [1, 5] as well as the *3D pentominoes problem* [6]:

– A *perfect squared 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 [7] and tested the corresponding 207 instances. The labelling strategy is roughly to repeat the following, first for the x dimension, then for the y dimension:
  1. Find the smallest position where some square can be placed.
  2. Find a square to place in that position.

– *Pentominoes* are pieces made of 5 connected unit cubes laid on a plane surface. Their shapes look like the 12 letters F, I, L, P, N, T, U, V, W, X, Y and Z. We considered the problem of finding the different ways of putting 12 distinct shapes that can be reflected and rotated in a box having a volume of 60 unit cubes. Our labelling strategy is roughly to repeat the following:
  1. Find a slot in the space that has not yet been filled by some piece.
  2. Find a piece that can fill that slot.

Fig. 6 (top right) and Table 1 respectively report, for the squared squares and the pentominoes problems, the time and number of backtracks for exploring all the search space[7] without breaking any symmetry. For the squared squares problems the maximum time of 1585 seconds was spent on problem 48; on the other hand, 148 problems were completely solved within 60 seconds. For the 3D pentomino packing instances, performance results for comparison can be found in [6]. However, they stop the search when the first 100 solutions have been found, so the results are only partly comparable.

Finally, wanting to compare the $geost$ kernel with a recent exact state of the art method for the 2D orthogonal packing problem [4], we reused the benchmarks proposed by Clautiaux et al. [8]. This is a feasibility problem which consists in determining whether a set of rectangles that cannot be rotated, can be packed or not into a rectangle of fixed size. In these instances the discrepancy between the sum of the areas of the rectangles to pack and the area of the big rectangle vary from 0% to 20%. We have 41 instances involving between 10 and 23 rectangles. Moreover, from these 41 instances, 26 instances are not feasible. In order to break symmetries between multiple rectangles of the same shape we added lexicographic ordering constraints. All x coordinates were labelled followed by all y coordinates, by decreasing rectangle size. Values were tried by increasing value. Fig. 6 (bottom) compares our results with the ones reported in [4]. Note that the sequence order for the curves differs, since the instances of each curve

[7] Finding all solutions and proving that there is no other solution.

are ordered by increasing y value. We solved all instances and are comparable with [4], although 8 instances are much easier for [4] and 10 instances are much easier for us.

Note that for the last three problems (i.e., Squared Squares, Pentominoes and 2D orthogonal packing) extra filtering algorithms mostly based on cumulative relaxation were integrated within our kernel. Since this report focusses on the constraint kernel and because of space limitations these methods were not detailed.

6 Related Work and Future Directions

The rectangles packing problem has been studied by Clautiaux et al. [8, 4] where scheduling-based reasoning is used [1]. The use of sweep algorithms in constraint filtering algorithms was introduced in [3] and applied to the non-overlapping 2D rectangles constraints. This report generalizes and extends that work in several ways.

– The 2D sweep is generalized to a lexicographic sweep, independent of the number of dimensions.
– The notion of forbidden regions for non-overlapping rectangles is generalized to necessary conditions for general geometric constraints.

The idea of generating necessary conditions is reminiscent of indexicals [9], a.k.a. projection constraints [10]. An indexical for a constraint $c(x_1, \ldots, x_n)$ computes a unary constraint on a single variable x_i, i.e. a set S of values such that $c \Rightarrow x_i \in S$, in reaction to domain changes in $x_1, \ldots, x_n$. The constraint kernel then immediately enforces $x_i \in S$. Our kernel generalizes this in two ways:

– We compute necessary conditions in the form of k-dimensional forbidden regions.
– We treat all such forbidden regions, for a selected object and shape, in one run of the sweep algorithm. Projecting a single forbidden region on one coordinate often does not yield any pruning, whereas considering the union of forbidden regions is much more effective.

Dal Palù et al. in [11] proposed a constraint solver specialized for 3D discrete domains. Their solver was targeted to the study of problems in molecular, chemical and crystal structures. Our work, however, remains in the setting of mainstream finite domain constraint systems, whereas our kernel internally handles k-dimensional objects.

Even though the $geost$ kernel has been designed over discrete domains, it could rather easily be extended to continuous domains with the coordinates of the objects approximated by the floating-point numbers $\mathbb{F}$. Since switching from $\mathbb{N}$ to $\mathbb{F}$ may cause rounding errors at this level, the sweep algorithm needs to handle these rounding errors when moving the sweep out of an outbox constraint. If the projections of the forbidden regions on all dimensions are intervals of real bounds we can proceed as follows. On continuous domains, an outbox will have an very thin strip at the border where the feasibility of the corresponding internal constraint is unknown. The region inside this strip is strictly forbidden, and outside, the constraints is certainly satisfied. The outbox must be computed including this strip, by taking lower and upper approximations of the region's coordinates. In that case, the solutions are guaranteed to be valid, but the solver may not be complete, because it may (rarely) happen that the real forbidden region allows positions that are forbidden by its approximation.

This research was conducted under the European Union project “Net-WMS”, a major task of which is to study packing problems in warehouse management. In this context, our constraint kernel is a step towards being able to capture a large set of packing rules in a constraint programming setting. Future work involves extending our set of external geometric constraints to include such packing rules.

7 Conclusion

The main contribution of this report is a geometrical constraint kernel for handling the location in space and time of polymorphic k-dimensional objects subject to various geometrical and time constraints. The constraint kernel is generic in the sense that one of its parameters is a set of constraints on subsets of the objects. These constraints are handled globally by the kernel.

We have presented a sweep algorithm for filtering the attributes of the objects. Thank to its architecture, new geometric constraints can be plugged into this sweep algorithm without modifying it. The strong point of this sweep algorithm is that it considers all the geometrical constraints for a selected object and shape in one run. As a first result, more deduction can be performed by combining sets of forbidden points coming from multiple geometrical constraints. Secondly, it can handle within one single constraint problems involving up several tens of thousands of objects without memory consumption problems, which is often a weak point for constraint programming environment. We have also shown that we could handle tight 2D or 3D placement problems, which were traditionally solved by specific approaches.

Acknowledgements

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

configuration	backtracks (1st)	time (1st)	backtracks (all)	time (all)	solutions
$20 \times 3 \times 1$	1434	1840	47381	49740	8
$15 \times 4 \times 1$	290	560	888060	939060	1472
$12 \times 5 \times 1$	1594	1850	3994455	4112870	4040
$10 \times 6 \times 1$	111	260	9688985	10726810	9356
$10 \times 3 \times 2$	1267	2370	1203511	1778980	96
$6 \times 5 \times 2$	157	730	n/a	n/a	n/a
$5 \times 4 \times 3$	3567	14930	n/a	n/a	n/a

Table 1. Performance evaluation. 3D pentomino packing instances. Time in milliseconds. “n/a” corresponds to a quantity that was not available with a time-out of several hours.

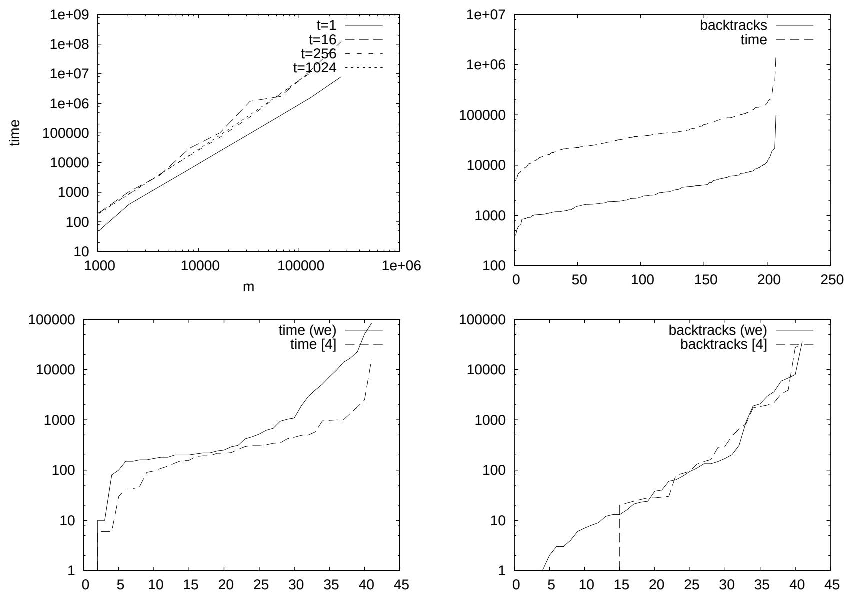


Fig. 6. Performance evaluation. Top left: scalability, $t \in \{1, 16, 256, 1024\}$. Top right: Perfect Squared Squares, runtime and backtracks. Bottom: 2D Orthogonal Packing, runtime (left) and backtracks (right). Time in milliseconds. In each curve, the instances are ordered by increasing y value.

8 Annex: Benchmark Data and Code

8.1 Random Instances

```
:- use_module(library(lists)).
:- use_module(library(random)).
:- use_module(library(clpfd)).

limit_rand(2, 10). % d=2:  li=rand(1..33)
limit_rand(3,  4). % d=3:  li=rand(1..11)
limit_rand(4,  3). % d=4:  li=rand(1.. 6)

% T is the number of types of objects (i.e., the number of shapes)
% K is the number of dimensions
% N is the number of objects
% generates a geost constraint with one non-overlapping constraint and search for a first solution

% a run with: top(1, 2, 1024). top(1, 2, 2048).   ... ...
% a run with: top(16, 2, 1024). top(16, 2, 2048).   ... ...
% a run with: top(256, 2, 1024). top(256, 2, 2048).   ... ...
% a run with: top(1024, 2, 1024). top(1024, 2, 2048).   ... ...

% a run with: top(1, 3, 1024). top(1, 3, 2048).   ... ...
% a run with: top(16, 3, 1024). top(16, 3, 2048).   ... ...
% a run with: top(256, 3, 1024). top(256, 3, 2048).   ... ...
% a run with: top(1024, 3, 1024). top(1024, 3, 2048).   ... ...

% a run with: top(1, 4, 1024). top(1, 4, 2048).   ... ...
% a run with: top(16, 4, 1024). top(16, 4, 2048).   ... ...
% a run with: top(256, 4, 1024). top(256, 4, 2048).   ... ...
% a run with: top(1024, 4, 1024). top(1024, 4, 2048).   ... ...

paper :-
    pow2(1024, N),
    t_param(T),
    k_param(K),
    top(T, K, N),
    fail.
paper.

t_param(1).
t_param(16).
t_param(256).
t_param(1024).

k_param(2).
k_param(3).
k_param(4).

pow2(P, P).
pow2(P, R) :-
    Q is P<<1,
    pow2(Q, R).


top(T, K, N) :-
    Goal = top(T, K, N),
    T1 is T+1,
    N1 is N+1,
    M1 is (N // T)+1, % number of objects plus one of a given type
    gen_ints(0, K, _Dimensions), % generates the list of dimensions
    gen_ints(1, N1, _ObjectsId), % generates the list of objects id
    length(Zeros, K), % generates a list with K ’0’ for the shifts
    domain(Zeros, 0, 0),
    gen_shps(1, T1, K, Zeros, Shapes1, Vol), % generates T random shapes
    sort_shapes(Shapes1, Shapes2),
    NeededVol is Vol*(M1-1),% overall volume of objects to place
    % place the origins of the objects in a k-dimensionsal box of size Limit
    % Limit is computed by majoring the needed volume by 20 percent and
    % by taking the 1/K root
    Limit is integer(floor(exp(NeededVol+((NeededVol*20)//100),1/K)))+5,
    gen_objects(1, T1, 1, M1, K, Limit, Objects, Variables),
    statistics(runtime, _),
    statistics_memory(Membase),
    (   diffn(Objects, Shapes2, [fixall(F)]),
        search_fixall(Variables, F),
        statistics(runtime, [_,T2]),
        statistics_memory(Mem),
        format('goal=~q CPU=~d memory=~d\n', [Goal,T2,Mem-Membase]) -> true
    ;   format('goal=~q failed\n', [Goal])
    ).

statistics_memory(Mem) :-
    garbage_collect,
    statistics(program, [P|_]),
    statistics(global_stack, [G|_]),
    statistics(local_stack, [L|_]),
    statistics(trail, [T|_]),
    statistics(choice, [C|_]),
    Mem is P+G+L+T+C.

sort_shapes(Shapes1, Shapes4) :-
    tag_shapes(Shapes1, Shapes2),
    keysort(Shapes2, Shapes3),
    rebuild_shapes(Shapes3, Shapes4, 0).

tag_shapes([], []).
tag_shapes([shape(_,Off,Size1)|S1], [Size2-Off|S2]) :-
    negate_shape(Size1, Size2),
    tag_shapes(S1, S2).

rebuild_shapes([], [], _).
rebuild_shapes([Size1-Off|S1], [shape(J,Off,Size2)|S2], I) :-
    J is I+1,
    negate_shape(Size1, Size2),
    rebuild_shapes(S1, S2, J).

negate_shape([], []).
negate_shape([X|Xs], [Y|Ys]) :-
    Y is -X,
    negate_shape(Xs, Ys).

search_fixall(_, 1).
search_fixall([], _).
search_fixall([X|Xs], F) :-
    indomain(X),
    search_fixall(Xs, F).

gen_objects(T, T, _, _, _, _, [], []) :- !.
gen_objects(J, T, Oid, M1, K, Limit, Objects, Variables) :-
    J < T,
    gen_objs(1, M1, Oid, J/*shape*/, K, Limit, Objs1, Vars1),
    J1 is J+1,
    NextOid is Oid+M1-1,
    gen_objects(J1, T, NextOid, M1, K, Limit, Objs2, Vars2),
    append(Objs1, Objs2, Objects),
    append(Vars1, Vars2, Variables).

gen_objs(M, M, _, _, _, _, [], []) :- !.
gen_objs(J, M, Oid, S, K, Limit, [object(Oid,S,Origins)|R], Variables) :-
    J < M,
    gen_origins(0, K, Limit, Origins),
    J1 is J+1,
    Oid1 is Oid+1,
    gen_objs(J1, M, Oid1, S, K, Limit, R, Vars),
    append(Origins, Vars, Variables).

gen_origins(K, K, _, []) :- !.
gen_origins(J, K, Limit, [O|R]) :-
    J < K,
    O in 1..Limit,
    J1 is J+1,
    gen_origins(J1, K, Limit, R).

gen_shps(M, M, _, _, [], 0) :- !.
gen_shps(J, M, K, Zeros, [shape(J,Zeros,Sizes)|R], Volum) :-
    J1 is J+1,
    limit_rand(K, Limit),
    gen_sizes(0, K, Limit, Sizes, Vol),
    gen_shps(J1, M, K, Zeros, R, Vol1),
    Volum is Vol+Vol1.

gen_sizes(K, K, _, [], 1) :- !.
gen_sizes(J, K, L, [S|R], V) :-
    J < K,
    random(1, L, S),
    J1 is J+1,
    gen_sizes(J1, K, L, R, V1),
    V is S*V1.

gen_ints(M, M, []) :- !.
gen_ints(J, M, [J|R]) :-
    J < M,
    J1 is J+1,
    gen_ints(J1, M, R).
```

8.2 Perfect Squared Squares

```
:- use_module(library(lists)).
:- use_module(library(clpfd)).

posted_domains :-
    findall(Size, posted_domains(_,Size), Sizes),
    sumlist(Sizes, Total),
    format('TOTAL domain size=~d\n', [Total]).

posted_domains(ID, Total) :-
    data(ID, _, Limit, Rev),
    reverse(Rev, Sizes),
    constraints(ID, diffn, Xs, _Ys, _Sizes, _Limit, [-c,-k], _),
    print_square_domains(Xs, Sizes, DomSizes),
    sumlist(DomSizes, Total),
    format('square instance=~w space=~d*~d total domain size=~d\n', [ID,Limit,Limit,Total]).

print_square_domains([], [], []).
print_square_domains([X|Xs], [S|Ss], [Sz|Szs]) :-
    fd_dom(X, Xd),
    fd_size(X, Sz),
    format('square ~dx~d, domain=~q, domain size=~d\n', [S,S,Xd,Sz]),
    print_square_domains(Xs, Ss, Szs).

runfirst(ID,Srch) :-
    Opt=[-c,-k],
    statistics(runtime, _),
    solve(ID, diffn, Srch, _, _, Opt),
    statistics(runtime, [_,Time]),
    fd_statistics(backtracks, B),
    logit('instance=~w opt=~w solutions=~d backtracks=~d time=~d\n', [ID,Opt,1,B,Time]).

runall(ID,Srch) :-
    Opt=[-c,-k],
    statistics(runtime, _),
    count(ID, diffn, Srch, N, Opt),
    statistics(runtime, [_,Time]),
    fd_statistics(backtracks, B),
    logit('instance=~w opt=~w solutions=~d backtracks=~d time=~d\n', [ID,Opt,N,B,Time]).

runall(Srch) :-
    retractall(btr_time/2),
    data(ID, _, _, _),
    statistics(runtime, _),
    count(ID, diffn, Srch, N, [-c,-k]),
    statistics(runtime, [_,Time]),
    fd_statistics(backtracks, Btr),
    logit('instance=~w opt=~w solutions=~d backtracks=~d time=~d\n', [ID,[-c,-k],N,Btr,Time]),
    assertz(btr_time(Btr,Time)),
    fail.
runall(_) :-
    findall(B-T, btr_time(B,T), Btr1),
    keysort(Btr1, Btr2),
    findall(T-B, btr_time(B,T), Time1),
    keysort(Time1, Time2),
    tell('squares.dat'),
    write_plot_data(Btr2, Time2, 0),
    told,
    keys_and_values(Time2, Times, Btrs),
    sumlist(Btrs, SumBtr),
    sumlist(Times, SumTime),
    format('TOTAL backtracks=~d time=~d\n', [SumBtr,SumTime]).

write_plot_data([], [], _).
write_plot_data([B-_|Bs], [T-_|Ts], I) :-
    J is I+1,
    format('~d ~d ~d\n', [J,B,T]),
    write_plot_data(Bs, Ts, J).

runfirst(Srch) :-
    Opt = [-c,-k],
    data(ID, _, _, _),
    (solve(ID, diffn, Srch, _, _, Opt) -> true),
    fd_statistics(backtracks, B),
    fd_statistics(resumptions, R),
    logit('instance=~w opt=~w backtracks=~d resumptions=~d\n', [ID,Opt,B,R]),
    fail.
runfirst(_).

logit(Fmt, Args) :-
    prolog_flag(argv, [Logfile]), !,
    open(Logfile, append, S),
    format(S, Fmt, Args),
    close(S).
logit(Fmt, Args) :-
    format(Fmt, Args).

count(ID, Type, Search, N, Opt) :-
    findall(1, solve(ID,Type,Search,_,_,Opt), L),
    length(L, N).

squares(ID, Type, Search, Opt) :-
    solve(ID, Type, Search, Xs, Ys, Opt),
    writeq(Xs), nl,
    writeq(Ys), nl, nl.

solve(ID, Type, Search, Xs, Ys, Opt) :-
    constraints(ID, Type, Xs, Ys, Sizes, Limit, Opt, 0),
    search(Search, Xs, Ys, Sizes, Limit, 0).

search(primal, Xs, Ys, _, _Limit, _Fixall) :-
    labeling([bisect], Xs),
    labeling([bisect], Ys).
search(dual, Xs, Ys, _, Limit, Fixall) :-
    dual_labeling(Xs, 1, Limit, Fixall),
    dual_labeling(Ys, 1, Limit, Fixall).
search(lex, Xs, Ys, Ss, Limit, _) :-
    transpose([Xs,Ys,Ss], Sqs),
    search_lex(Sqs, [], [1,1], Limit).

search_lex([], _, _, _) :- !.
search_lex(Sqs, Done, P, Limit) :-
    member(Pl, Done),
    in_square(P, Pl, P1, Limit), !,
    search_lex(Sqs, Done, P1, Limit).
search_lex(Sqs, Done, P, Limit) :-
    search_lex2(Sqs, Sqs1, Done, Done1, P, P1, Limit),
    search_lex(Sqs1, Done1, P1, Limit).

search_lex2([Sq|Sqs], Sqs, Done, [Sq|Done], P, P1, Limit) :-
    assign2(Sq, P),
    in_square(P, Sq, P1, Limit).
search_lex2([Sq|Sqs], [Sq|Sqs1], Pl, Pl1, P, P1, Limit) :-
    Sq = [X,Y,_],
    lex_chain([P,[X,Y]], [op(#<)]),
    search_lex2(Sqs, Sqs1, Pl, Pl1, P, P1, Limit).

assign2([X,Y,_], [U,V]) :-
    clpfd:'$fd_in_interval'(X, U, U, 1),
    clpfd:'$fd_in_interval'(Y, V, V, 0),
    clpfd:'$fd_evaluate_indexical'(RC, Global),
    clpfd:evaluate(RC, Global).

in_square(P, [X,Y,S], P1, Limit) :-
    P = [Px,Py],
    X1 is X+S-1,
    Y1 is Y+S-1,
    Px in X..X1,
    Py in Y..Y1,
    (   Y1=:=Limit ->
        Px1 is Px+1,
        P1 = [Px1,1]
    ;   Y2 is Y1+1,
        P1 = [Px,Y2]
    ).

dual_labeling(_,  _, _, 1) :- !.
dual_labeling([], _, _, _) :- !.
dual_labeling(L, I, Limit, Fixall) :-
    dual_labeling(L, L1, I, Limit, J, Fixall),
    dual_labeling(L1, J, Limit, Fixall).

dual_labeling([], [], _, J, J, _) :- !.
dual_labeling([X|L1], L2, I, J0, J, Fixall) :-
    (   integer(X) -> dual_labeling(L1, L2, I, J0, J, Fixall)
    ;   X #= I, dual_labeling(L1, L2, I, J0, J, Fixall)
    ;   X #> I,
        fd_min(X, J1),
        J2 is min(J0,J1),
        L2 = [X|L3],
        dual_labeling(L1, L3, I, J2, J, Fixall)
    ).

constraints(ID, Type, Xs, Ys, Sizes, Limit, Opt, Fixall) :-
    generate_squares(ID, Xs, Ys, Sizes, Limit),
    order_squares(Xs, Ys, Sizes),
    % state_asymmetry(Xs, Ys, Sizes, Limit),
    state_no_overlap(Type, Xs, Ys, Sizes, Fixall),
    (   memberchk(-c, Opt) -> true
    ;   cumulative(Xs, Sizes, Sizes, Limit),
        cumulative(Ys, Sizes, Sizes, Limit)
    ),
    (   memberchk(-k, Opt) -> true
    ;   knapsack(0, Limit, Xs, Sizes),
        knapsack(0, Limit, Ys, Sizes)
    ),
    true.

knapsack(L, L, _, _) :- !.
knapsack(I, L, Xs, Sizes) :-
    J is I+1,
    crossings(Xs, Sizes, Bs, J),
    scalar_product(Sizes, Bs, #=, L, [consistency(domain)]),
    knapsack(J, L, Xs, Sizes).

crossings([], [], [], _).
crossings([X|Xs], [S|Ss], [B|Bs], J) :-
    J0 is J-S+1,
    X in J0..J #<=> B,
    crossings(Xs, Ss, Bs, J).

% for compatibility
cumulative(Os, Ds, Hs, L) :-
    mktasks(Os, Ds, Hs, Tasks),
    cumulative(Tasks, [limit(L),global(true)]).

mktasks([], [], [], []).
mktasks([O|Os], [D|Ds], [H|Hs], [task(O,D,E,H,0)|Tasks]) :-
    E in 0..1000,
    mktasks(Os, Ds, Hs, Tasks).

generate_squares(ID, Xs, Ys, Sizes, Size) :-
    data(ID, _, Size, Rev),
    reverse(Rev, Sizes),
    generate_coordinates(Xs, Ys, Sizes, Size).

generate_coordinates([], [], [], _).
generate_coordinates([X|Xs], [Y|Ys], [S|Ss], Size) :-
    Sd is Size-S+1,
    domain([X,Y], 1, Sd),
    generate_coordinates(Xs, Ys, Ss, Size).

order_squares(Xs, Ys, Sizes) :-
    sxy(Sizes, Xs, Ys, L1),
    keysort(L1, L2),
    keyclumped(L2, L3),
    order_groups(L3).

sxy([], [], [], []).
sxy([S|Ss], [X|Xs], [Y|Ys], [S-[X,Y]|Zs]) :-
    sxy(Ss, Xs, Ys, Zs).

order_groups([]).
order_groups([_-L|Groups]) :-
    lex_chain(L, [op(#<)]),
    order_groups(Groups).

% first square has center in SW quarter, under the positive diagonal
state_asymmetry([X|_], [Y|_], [D|_], Limit) :-
    UB is (Limit-D+2)>>1,
    X in 1..UB,
    Y #=< X.

state_no_overlap(disjoint, Xs, Ys, Sizes, _) :- !,
    disjoint_data(Xs, Ys, Sizes, Data),
    disjoint2(Data).
state_no_overlap(diffn, Xs, Ys, Sizes, Fixall) :- !,
    diffn_data(Xs, Ys, Sizes, Objs, Shapes, 1),
    diffn(Objs, Shapes, [fixall(Fixall)]).

state_no_overlap(_, [], [], []) :- !.
state_no_overlap(Type, [X|Xs], [Y|Ys], [S|Ss]) :-
    state_no_overlap(Type, X, Y, S, Xs, Ys, Ss),
    state_no_overlap(Type, Xs, Ys, Ss).

disjoint_data([], [], [], []).
disjoint_data([X|Xs], [Y|Ys], [S|Ss], [r(X,S,Y,S)|Rs]) :-
    disjoint_data(Xs, Ys, Ss, Rs).

diffn_data([], [], [], [], [], _).
diffn_data([X|Xs], [Y|Ys], [S|Ss], [object(I,I,[X,Y])|Os], [shape(I,[0,0],[S,S])|Hs], I) :-
    J is I+1,
    diffn_data(Xs, Ys, Ss, Os, Hs, J).

state_no_overlap(_, _, _, _, [], [], []) :- !.
state_no_overlap(Type, X, Y, S, [X1|Xs], [Y1|Ys], [S1|Ss]) :-
    no_overlap(Type, X, Y, S, X1, Y1, S1),
    state_no_overlap(Type, X, Y, S, Xs, Ys, Ss).

no_overlap(spec, X1, _Y1, S1, X2, _Y2, _S2) :-
    leqc(X1, S1, X2).
no_overlap(spec, X1, _Y1, _S1, X2, _Y2, S2) :-
    leqc(X2, S2, X1).
no_overlap(spec, _X1, Y1, S1, _X2, Y2, _S2) :-
    leqc(Y1, S1, Y2).
no_overlap(spec, _X1, Y1, _S1, _X2, Y2, S2) :-
    leqc(Y2, S2, Y1).
no_overlap(card, X1, Y1, S1, X2, Y2, S2) :-
    X1+S1 #=< X2 #<=> B1,
    X2+S2 #=< X1 #<=> B2,
    Y1+S1 #=< Y2 #<=> B3,
    Y2+S2 #=< Y1 #<=> B4,
    B1+B2+B3+B4 #>= 1.
no_overlap(wcd, X1, Y1, S1, X2, Y2, S2) :-
    no_overlap_ix(X1, Y1, S1, X2, Y2, S2).

leqc(X1, S1, X2) :- X1+S1 #=< X2.

no_overlap_ix(X1, Y1, S1, X2, Y2, S2) +:
    X1 in ((((min(Y1)+S1)..max(Y2))\/((min(Y2)+S2)..max(Y1))) ? (inf..sup))
         \/ \(max(X2)-(S1-1) .. min(X2)+(S2-1)),
    X2 in ((((min(Y1)+S1)..max(Y2))\/((min(Y2)+S2)..max(Y1))) ? (inf..sup))
         \/ \(max(X1)-(S2-1) .. min(X1)+(S1-1)),
    Y1 in ((((min(X1)+S1)..max(X2))\/((min(X2)+S2)..max(X1))) ? (inf..sup))
         \/ \(max(Y2)-(S1-1) .. min(Y2)+(S2-1)),
    Y2 in ((((min(X1)+S1)..max(X2))\/((min(X2)+S2)..max(X1))) ? (inf..sup))
         \/ \(max(Y1)-(S2-1) .. min(Y1)+(S1-1)).

:- dynamic data/4.
data(id001,21,112,[2,4,6,7,8,9,11,15,16,17,18,19,24,25,27,29,33,35,37,42,50]).
data(id002,22,110,[2,3,4,6,7,8,12,13,14,15,16,17,18,21,22,23,24,26,27,28,50,60]).
data(id003,22,110,[1,2,3,4,6,8,9,12,14,16,17,18,19,21,22,23,24,26,27,28,50,60]).
data(id004,22,139,[1,2,3,4,7,8,10,17,18,20,21,22,24,27,28,29,30,31,32,38,59,80]).
data(id005,22,147,[1,3,4,5,8,9,17,20,21,23,25,26,29,31,32,40,43,44,47,48,52,55]).
data(id006,22,147,[2,4,8,10,11,12,15,19,21,22,23,25,26,32,34,37,41,43,45,47,55,59]).
data(id007,22,154,[2,5,9,11,16,17,19,21,22,24,26,30,31,33,35,36,41,46,47,50,52,61]).
data(id008,22,172,[1,2,3,4,9,11,13,16,17,18,19,22,24,33,36,38,39,42,44,53,75,97]).
data(id009,22,192,[4,8,9,10,12,14,17,19,26,28,31,35,36,37,41,47,49,57,59,62,71,86]).
data(id010,23,110,[1,2,3,4,5,7,8,10,12,13,14,15,16,19,21,28,29,31,32,37,38,41,44]).
data(id011,23,139,[1,2,7,8,12,13,14,15,16,18,19,20,21,22,24,26,27,28,32,33,38,59,80]).
data(id012,23,140,[1,2,3,4,5,8,10,13,16,19,20,23,27,28,29,31,33,38,42,45,48,53,54]).
data(id013,23,140,[2,3,4,7,8,9,12,15,16,18,22,23,24,26,28,30,33,36,43,44,47,50,60]).
data(id014,23,145,[1,2,3,4,6,8,9,12,15,20,22,24,25,26,27,29,30,31,32,34,36,61,84]).
data(id015,23,180,[2,4,8,10,11,12,15,19,21,22,23,25,26,32,33,34,37,41,43,45,47,88,92]).
data(id016,23,188,[2,4,8,10,11,12,15,19,21,22,23,25,26,32,33,34,37,45,47,49,51,92,96]).
data(id017,23,208,[1,3,4,9,10,11,12,16,17,18,22,23,24,40,41,60,62,65,67,70,71,73,75]).
data(id018,23,215,[1,3,4,9,10,11,12,16,17,18,22,23,24,40,41,60,66,68,70,71,74,76,79]).
data(id019,23,228,[2,7,9,10,15,16,17,18,22,23,25,28,36,39,42,56,57,68,69,72,73,87,99]).
data(id020,23,257,[2,3,9,11,14,15,17,20,22,24,28,29,32,33,49,55,57,60,63,66,79,123,134]).
data(id021,23,332,[1,15,17,24,26,30,31,38,47,48,49,50,53,56,58,68,83,89,91,112,120,123,129]).
data(id022,24,120,[3,4,5,6,8,9,10,12,13,14,15,16,17,19,20,23,25,32,33,34,40,41,46,47]).
data(id023,24,186,[2,3,4,7,8,9,12,15,16,18,22,23,24,26,28,30,33,36,43,46,47,60,90,96]).
data(id024,24,194,[2,3,7,9,10,16,17,18,19,20,23,25,28,34,36,37,42,53,54,61,65,68,69,72]).
data(id025,24,195,[2,4,7,10,11,16,17,18,21,26,27,30,39,41,42,45,47,49,52,53,54,61,63,80]).
data(id026,24,196,[1,2,5,10,11,15,17,18,20,21,24,26,29,31,32,34,36,40,44,47,48,51,91,105]).
data(id027,24,201,[1,3,4,6,9,10,11,12,17,18,20,21,22,23,26,38,40,46,50,52,53,58,98,103]).
data(id028,24,201,[1,4,5,8,9,10,11,15,16,18,19,20,22,24,26,39,42,44,49,52,54,56,93,108]).
data(id029,24,203,[1,2,5,10,11,15,17,18,20,21,24,26,29,31,32,34,36,40,44,48,54,58,98,105]).
data(id030,24,247,[3,5,6,9,12,14,19,23,24,25,28,32,34,36,40,45,46,48,56,62,63,66,111,136]).
data(id031,24,253,[2,4,5,9,13,18,20,23,24,27,28,31,38,40,44,50,61,70,72,77,79,86,88,104]).
data(id032,24,255,[3,5,10,11,16,17,20,22,23,25,26,27,28,32,41,44,52,53,59,63,65,74,118,137]).
data(id033,24,288,[2,7,9,10,15,16,17,18,22,23,25,28,36,39,42,56,57,60,68,72,73,87,129,159]).
data(id034,24,288,[1,5,7,8,9,14,17,20,21,26,30,32,34,36,48,51,54,59,64,69,72,93,123,165]).
data(id035,24,290,[2,3,8,9,11,12,14,17,21,30,31,33,40,42,45,48,59,61,63,65,82,84,124,166]).
data(id036,24,292,[1,2,3,8,12,15,16,17,20,22,24,26,29,33,44,54,57,60,63,67,73,102,117,175]).
data(id037,24,304,[3,5,7,11,12,17,20,22,25,29,35,47,48,55,56,57,69,72,76,92,96,100,116,132]).
data(id038,24,304,[3,4,7,12,16,20,23,24,27,28,30,32,33,36,37,44,53,57,72,76,85,99,129,175]).
data(id039,24,314,[2,4,11,12,16,17,18,19,28,29,40,44,47,59,62,64,65,78,79,96,97,105,113,139]).
data(id040,24,316,[3,9,10,12,13,14,15,23,24,33,36,37,48,52,54,55,57,65,66,78,79,93,144,172]).
data(id041,24,326,[1,6,10,11,14,15,18,24,29,32,43,44,53,56,63,65,71,80,83,101,104,106,119,142]).
data(id042,24,423,[2,9,15,17,27,29,31,32,33,36,47,49,50,60,62,77,105,114,123,127,128,132,168,186]).
data(id043,24,435,[1,2,8,10,13,19,23,33,44,45,56,74,76,78,80,88,93,100,112,131,142,143,150,192]).
data(id044,24,435,[3,5,9,11,12,21,24,27,30,44,45,50,54,55,63,95,101,112,117,123,134,140,178,200]).
data(id045,24,459,[8,9,10,11,16,30,36,38,45,55,57,65,68,84,95,98,100,116,117,126,135,144,180,198]).
data(id046,24,459,[4,6,9,10,17,21,23,25,31,33,36,38,45,50,83,115,117,126,133,135,144,146,180,198]).
data(id047,24,479,[5,6,17,23,24,26,28,29,35,43,44,52,60,68,77,86,130,140,150,155,160,164,174,175]).
data(id048,25,147,[3,4,5,6,8,9,10,12,13,14,15,16,17,19,20,23,25,27,32,33,34,40,41,73,74]).
data(id049,25,208,[1,2,3,4,5,7,8,11,12,17,18,24,26,28,29,30,36,39,44,45,50,59,60,89,119]).
data(id050,25,213,[3,5,6,7,13,16,17,20,21,23,24,25,26,28,31,35,36,47,49,56,58,74,76,81,90]).
data(id051,25,215,[1,4,6,7,11,15,24,26,27,33,37,39,40,41,42,43,45,47,51,55,60,62,63,69,83]).
data(id052,25,216,[1,2,3,4,5,7,8,11,16,17,18,19,25,30,32,33,39,41,45,49,54,59,64,103,113]).
data(id053,25,236,[1,2,4,9,11,12,13,14,15,16,19,24,38,40,44,46,47,48,59,64,65,70,81,85,107]).
data(id054,25,242,[1,3,6,7,9,13,14,16,17,19,23,25,26,28,30,31,47,51,54,57,60,64,67,111,131]).
data(id055,25,244,[1,2,4,5,7,10,15,17,19,20,21,22,26,27,30,37,40,41,45,65,66,68,70,110,134]).
data(id056,25,252,[4,7,10,11,12,13,23,25,29,31,32,34,36,37,38,40,42,44,62,67,68,71,77,108,113]).
data(id057,25,253,[2,4,5,6,9,10,12,14,20,24,27,35,36,37,38,42,43,45,50,54,63,66,70,120,133]).
data(id058,25,260,[1,4,6,7,10,15,24,26,27,28,29,31,33,34,37,38,44,65,70,71,77,78,83,100,112]).
data(id059,25,264,[3,7,8,12,16,18,19,20,22,24,26,31,34,37,38,40,42,53,54,61,64,69,70,130,134]).
data(id060,25,264,[3,8,12,13,16,18,20,21,22,24,26,29,34,38,40,42,43,47,54,59,64,70,71,130,134]).
data(id061,25,264,[1,3,4,6,9,10,11,12,16,17,18,20,21,22,39,42,54,56,61,66,68,69,73,129,135]).
data(id062,25,265,[1,3,4,6,9,10,11,12,16,17,18,20,21,22,39,42,54,56,62,66,68,69,74,130,135]).
data(id063,25,273,[1,4,8,10,11,12,17,19,21,22,27,29,30,33,37,43,52,62,65,86,88,89,91,96,120]).
data(id064,25,273,[1,6,9,14,16,17,18,21,22,23,25,31,32,38,44,46,48,50,54,62,65,68,78,133,140]).
data(id065,25,275,[2,3,7,13,17,24,25,31,33,34,35,37,41,49,51,53,55,60,68,71,74,81,94,100,107]).
data(id066,25,276,[1,5,8,9,11,18,19,21,30,36,41,44,45,46,47,51,53,58,63,69,71,84,87,105,120]).
data(id067,25,280,[5,6,11,17,18,20,21,24,27,28,32,34,41,42,50,53,54,55,68,78,85,88,95,97,117]).
data(id068,25,280,[2,3,7,8,14,18,30,36,37,39,44,50,52,54,56,60,63,64,65,72,75,78,79,96,106]).
data(id069,25,284,[1,2,11,12,14,16,18,19,23,26,29,37,38,39,40,42,59,68,69,77,78,97,106,109,110]).
data(id070,25,286,[1,4,5,7,10,12,15,16,20,23,28,30,32,33,35,37,53,54,64,68,74,79,80,133,153]).
data(id071,25,289,[2,3,5,8,13,14,17,20,21,32,36,41,50,52,60,61,62,68,74,76,83,87,100,102,104]).
data(id072,25,289,[2,3,4,5,7,12,16,17,19,21,23,25,29,31,32,44,57,64,65,68,72,76,84,140,149]).
data(id073,25,290,[1,2,10,11,13,14,15,17,18,28,29,34,36,38,50,56,60,69,77,80,85,91,94,111,119]).
data(id074,25,293,[5,6,11,17,18,20,21,24,27,28,32,34,41,42,50,54,55,66,68,78,85,88,95,110,130]).
data(id075,25,297,[2,7,8,9,10,15,16,17,18,23,25,26,28,36,38,43,53,60,61,68,69,77,99,137,160]).
data(id076,25,308,[1,3,4,7,10,12,13,23,25,34,37,38,39,43,44,45,62,77,79,85,87,108,113,115,116]).
data(id077,25,308,[1,5,6,7,8,9,13,16,19,28,33,36,38,43,45,48,70,71,73,84,86,102,104,120,133]).
data(id078,25,309,[7,8,14,16,23,24,25,26,31,33,34,39,48,56,59,60,62,70,76,82,92,100,101,108,117]).
data(id079,25,311,[2,7,8,9,10,15,16,17,18,23,25,26,28,36,38,43,53,60,61,68,83,91,99,151,160]).
data(id080,25,314,[1,6,7,11,16,22,26,29,32,36,38,44,51,53,64,69,70,73,74,75,85,87,101,116,128]).
data(id081,25,316,[1,3,9,12,21,26,30,33,34,35,38,39,40,41,53,56,59,69,79,85,96,103,111,117,120]).
data(id082,25,317,[1,5,6,7,8,9,16,17,19,32,37,40,42,47,49,52,59,75,81,92,94,110,112,113,126]).
data(id083,25,320,[2,7,8,9,12,14,15,21,23,35,38,44,46,49,53,54,56,63,96,101,103,105,108,112,116]).
data(id084,25,320,[3,8,9,11,17,18,22,25,26,27,29,30,31,33,35,49,51,67,72,73,80,85,95,152,168]).
data(id085,25,320,[1,4,6,7,8,13,14,16,24,28,30,33,34,38,41,42,57,60,69,78,81,90,92,150,170]).
data(id086,25,320,[3,4,6,8,9,14,15,16,24,28,30,31,34,38,39,42,59,60,71,78,79,90,92,150,170]).
data(id087,25,322,[3,4,8,9,10,16,18,20,22,23,24,28,31,38,44,47,64,65,68,76,80,81,97,144,178]).
data(id088,25,322,[3,4,8,10,15,16,18,19,20,22,24,28,35,38,44,53,59,64,68,76,80,85,93,144,178]).
data(id089,25,323,[2,3,4,7,10,13,15,18,23,32,34,35,36,42,46,50,57,60,66,72,78,87,98,159,164]).
data(id090,25,323,[3,8,9,11,17,18,22,25,26,27,29,30,31,33,35,49,51,67,72,73,83,88,95,155,168]).
data(id091,25,323,[2,6,9,11,13,14,18,19,20,23,27,28,29,42,46,48,60,64,72,74,79,82,98,146,177]).
data(id092,25,325,[3,5,6,11,12,13,18,23,25,28,32,37,40,43,45,46,51,79,92,99,103,108,112,114,134]).
data(id093,25,326,[1,4,8,10,12,16,21,22,24,27,28,35,36,37,38,46,49,68,70,75,88,90,93,158,168]).
data(id094,25,327,[2,9,10,12,13,16,19,21,23,26,36,44,46,52,55,61,62,74,84,87,100,103,104,120,140]).
data(id095,25,328,[2,3,4,7,8,10,14,17,26,27,28,36,38,40,42,45,53,58,73,74,79,94,102,152,176]).
data(id096,25,334,[1,4,8,10,12,16,21,22,24,27,28,35,36,37,38,46,49,68,75,78,88,93,98,166,168]).
data(id097,25,336,[2,3,4,7,8,10,14,17,26,27,28,36,38,40,45,50,53,58,73,74,79,94,110,152,184]).
data(id098,25,338,[1,4,8,10,12,16,19,22,24,25,28,36,37,38,39,46,53,68,70,73,94,96,101,164,174]).
data(id099,25,338,[4,5,8,10,12,15,16,21,22,24,28,33,36,38,43,46,57,68,70,77,94,96,97,164,174]).
data(id100,25,340,[1,4,5,6,11,13,16,17,22,24,44,46,50,51,52,53,61,64,66,79,84,85,92,169,171]).
data(id101,25,344,[2,3,8,11,14,17,19,21,23,25,27,36,39,44,48,53,56,71,77,83,86,89,98,169,175]).
data(id102,25,359,[7,8,9,10,14,17,18,23,25,27,29,31,40,41,43,46,69,74,82,85,90,98,102,172,187]).
data(id103,25,361,[2,6,7,8,9,14,20,22,26,27,32,34,36,47,49,56,66,67,74,82,89,98,107,156,205]).
data(id104,25,363,[1,4,6,12,13,20,21,25,26,27,28,32,37,41,45,53,58,64,69,91,97,102,106,155,208]).
data(id105,25,364,[2,3,4,6,8,9,13,14,16,19,23,24,28,29,52,57,64,75,82,91,98,100,109,173,191]).
data(id106,25,367,[1,4,6,12,13,20,21,25,26,27,28,32,37,41,49,53,58,64,69,91,97,102,110,155,212]).
data(id107,25,368,[1,6,15,16,17,18,22,25,31,33,39,42,45,46,47,48,51,69,72,88,91,96,112,160,208]).
data(id108,25,371,[1,2,7,8,20,21,22,24,26,28,30,38,43,46,50,51,64,65,70,90,95,102,109,160,211]).
data(id109,25,373,[3,6,7,8,15,17,22,23,31,32,35,41,43,60,62,68,79,87,104,105,114,120,121,138,148]).
data(id110,25,378,[2,3,10,17,18,20,21,22,24,27,31,38,41,48,51,56,68,78,80,85,87,96,117,165,213]).
data(id111,25,378,[1,2,7,13,15,17,18,25,27,29,30,31,42,43,46,56,61,68,73,93,100,105,112,161,217]).
data(id112,25,380,[4,7,17,18,19,20,21,26,31,33,35,40,45,48,49,60,67,73,79,81,87,107,113,186,194]).
data(id113,25,380,[4,5,6,9,13,15,16,17,22,24,33,38,44,49,50,56,60,67,82,84,95,108,121,177,203]).
data(id114,25,381,[12,13,21,23,25,27,35,36,42,45,54,57,59,60,79,82,84,85,92,95,96,100,110,111,186]).
data(id115,25,384,[1,4,8,9,11,12,19,21,27,32,35,44,45,46,47,51,60,67,84,89,96,108,120,180,204]).
data(id116,25,384,[1,4,8,9,11,12,15,17,19,25,26,31,32,37,44,57,60,81,84,96,99,108,120,180,204]).
data(id117,25,384,[3,5,7,11,12,17,20,22,25,29,35,47,48,55,56,57,69,72,76,80,96,100,116,172,212]).
data(id118,25,385,[1,2,7,13,15,17,18,25,27,29,30,31,43,46,49,56,61,68,73,93,100,105,119,161,224]).
data(id119,25,392,[4,7,8,15,23,26,29,30,31,32,34,43,48,55,56,68,77,88,98,106,116,135,141,151,153]).
data(id120,25,392,[10,12,14,16,19,21,25,27,31,35,39,41,51,52,54,55,73,92,98,115,121,123,129,148,171]).
data(id121,25,392,[1,4,5,8,11,14,16,21,22,24,27,28,30,31,52,64,81,83,96,97,98,99,114,195,197]).
data(id122,25,393,[4,8,16,20,23,24,25,27,29,37,44,45,50,53,64,66,68,69,73,85,91,101,116,186,207]).
data(id123,25,396,[1,4,5,14,16,32,35,36,46,47,48,49,68,69,73,93,94,97,99,104,110,111,125,126,160]).
data(id124,25,396,[1,4,5,8,11,14,16,21,22,24,27,28,30,31,52,64,81,83,98,99,100,101,114,197,199]).
data(id125,25,396,[3,8,9,11,14,16,17,18,31,32,41,45,48,56,60,66,73,75,81,82,98,99,117,180,216]).
data(id126,25,398,[2,6,7,11,15,17,23,28,29,39,44,46,53,56,58,65,68,99,100,119,120,134,144,145,154]).
data(id127,25,400,[3,6,21,23,24,26,29,35,37,40,41,47,53,55,64,76,79,81,99,100,121,122,137,142,179]).
data(id128,25,404,[3,6,7,14,17,20,21,26,28,31,32,39,46,53,54,68,71,80,88,92,100,111,113,199,205]).
data(id129,25,404,[4,7,10,11,12,13,16,18,20,23,25,28,29,32,47,62,70,88,93,96,101,114,127,189,215]).
data(id130,25,408,[2,3,7,13,16,18,20,27,30,33,41,43,46,52,54,57,72,79,84,100,105,108,116,195,213]).
data(id131,25,412,[3,11,12,15,21,26,32,39,43,47,54,60,68,73,83,85,86,87,89,99,114,129,139,144,169]).
data(id132,25,413,[5,7,17,20,34,38,39,48,56,57,59,60,64,65,70,72,75,81,105,106,110,125,148,153,155]).
data(id133,25,416,[2,4,7,11,13,24,25,30,35,37,39,40,44,58,62,65,82,104,112,120,128,135,143,153,169]).
data(id134,25,416,[1,2,3,8,12,15,16,17,20,22,24,26,29,31,64,75,85,88,91,94,98,104,133,179,237]).
data(id135,25,421,[1,2,4,5,7,9,12,16,20,22,23,35,38,48,56,83,94,104,116,118,128,140,150,153,177]).
data(id136,25,421,[5,11,12,17,18,20,23,26,29,36,38,40,44,51,55,59,72,92,97,102,105,107,117,199,222]).
data(id137,25,422,[2,4,7,13,16,18,20,23,28,29,38,43,46,51,59,68,74,79,86,93,100,111,132,179,243]).
data(id138,25,425,[3,4,5,9,10,12,13,14,16,19,20,31,46,48,56,79,102,104,116,126,128,140,142,157,181]).
data(id139,25,441,[5,6,7,16,18,23,24,27,38,39,47,51,52,62,66,72,80,84,92,101,102,118,120,219,222]).
data(id140,25,454,[1,2,11,17,29,34,35,46,48,51,53,55,63,69,79,87,88,91,109,134,136,143,150,161,184]).
data(id141,25,456,[5,7,10,11,13,15,18,19,31,49,50,52,59,60,63,72,77,115,128,129,135,142,148,179,193]).
data(id142,25,465,[6,9,13,14,19,21,24,25,31,32,53,56,64,73,74,82,91,111,125,127,137,139,153,173,201]).
data(id143,25,472,[7,9,13,15,26,34,35,44,47,51,58,61,65,81,87,103,104,115,118,123,128,133,136,148,221]).
data(id144,25,477,[3,5,12,16,19,22,25,26,37,41,49,72,76,77,82,86,87,115,117,135,141,149,167,169,193]).
data(id145,25,492,[2,9,15,17,27,29,31,32,33,36,47,49,50,60,62,69,77,105,114,123,127,128,132,237,255]).
data(id146,25,492,[3,5,9,11,12,21,24,27,30,44,45,50,54,55,57,63,95,101,112,117,123,134,140,235,257]).
data(id147,25,503,[4,15,16,19,22,23,25,27,33,34,50,62,67,87,88,93,100,113,135,143,149,157,167,179,211]).
data(id148,25,506,[1,7,24,26,33,35,40,45,47,51,55,69,87,90,93,96,117,125,134,145,146,147,160,162,199]).
data(id149,25,507,[2,3,7,11,13,15,28,34,43,50,57,64,80,83,86,89,107,115,116,127,149,163,175,183,217]).
data(id150,25,512,[1,7,8,9,10,15,22,32,34,46,51,65,69,71,91,105,109,111,136,139,152,157,173,200,203]).
data(id151,25,512,[1,6,7,8,9,13,17,19,35,45,47,57,62,73,88,93,104,107,128,130,151,163,184,198,221]).
data(id152,25,513,[6,9,10,17,19,24,28,29,37,39,64,65,68,81,98,99,102,115,145,147,153,159,165,189,201]).
data(id153,25,517,[5,6,7,16,20,24,28,33,38,43,63,71,80,83,86,92,98,122,132,148,164,166,173,180,205]).
data(id154,25,524,[9,12,20,21,33,35,37,39,54,55,61,62,87,90,98,101,125,132,135,141,145,159,163,164,220]).
data(id155,25,527,[11,12,13,14,19,30,41,47,50,52,59,68,71,81,94,97,107,132,147,151,155,169,175,183,197]).
data(id156,25,528,[2,9,15,17,27,29,31,32,33,36,47,49,50,60,62,69,77,123,127,128,132,141,150,255,273]).
data(id157,25,529,[9,12,20,21,33,35,37,39,54,55,61,62,87,90,98,101,125,132,140,141,145,159,163,169,225]).
data(id158,25,531,[6,9,10,17,19,24,29,31,39,40,67,68,71,84,101,102,105,118,151,153,159,165,171,195,207]).
data(id159,25,532,[16,18,26,27,33,39,41,50,51,55,69,71,84,87,91,94,132,133,141,143,164,168,169,173,195]).
data(id160,25,534,[11,13,15,17,18,27,38,44,49,52,60,61,68,81,87,94,107,135,149,153,159,171,174,189,210]).
data(id161,25,535,[2,8,26,27,36,41,45,57,62,77,88,95,97,99,101,102,109,114,117,118,141,147,168,192,226]).
data(id162,25,536,[1,8,21,30,31,32,33,41,44,46,49,55,57,61,84,91,113,134,137,139,150,155,176,205,247]).
data(id163,25,536,[3,5,9,11,12,21,24,27,30,44,45,50,54,55,57,63,95,117,123,134,140,145,156,257,279]).
data(id164,25,540,[1,7,8,9,10,14,19,34,36,51,58,69,81,83,97,109,111,115,136,149,152,167,183,208,221]).
data(id165,25,540,[6,13,15,25,28,36,43,47,55,57,58,59,60,65,82,89,91,107,124,127,144,163,183,233,250]).
data(id166,25,540,[8,9,10,11,16,30,36,38,45,55,57,65,68,81,84,95,98,100,116,117,126,135,144,261,279]).
data(id167,25,540,[8,9,10,11,16,30,36,38,45,55,57,65,68,81,84,95,98,100,116,117,126,135,144,261,279]).
data(id168,25,540,[4,6,9,10,17,21,23,25,31,33,36,38,45,50,81,83,115,117,126,133,135,144,146,261,279]).
data(id169,25,540,[4,6,9,10,17,21,23,25,31,33,36,38,45,50,81,83,115,117,126,133,135,144,146,261,279]).
data(id170,25,541,[3,4,11,13,16,17,21,25,26,44,46,64,75,86,87,97,106,109,133,141,165,185,191,215,217]).
data(id171,25,541,[3,5,27,32,33,37,47,50,53,56,57,69,71,78,97,98,109,111,126,144,165,169,183,189,232]).
data(id172,25,544,[1,7,24,26,33,35,40,45,47,51,55,69,87,90,93,96,117,125,134,145,147,184,198,199,200]).
data(id173,25,544,[6,8,20,21,23,41,42,48,59,61,77,80,81,85,90,92,93,102,115,132,139,168,198,207,244]).
data(id174,25,547,[3,5,16,22,26,27,35,47,49,59,67,71,72,85,87,102,103,111,137,144,150,197,200,203,207]).
data(id175,25,549,[4,10,14,24,26,31,34,36,38,40,43,48,59,63,74,89,97,105,117,124,136,152,156,241,308]).
data(id176,25,550,[1,2,5,13,19,20,25,30,39,43,58,59,73,75,76,90,95,103,116,128,130,132,172,262,288]).
data(id177,25,550,[1,11,16,23,24,27,29,36,41,43,44,47,59,70,71,80,99,103,111,116,128,156,167,227,323]).
data(id178,25,551,[3,5,24,25,26,30,35,36,39,40,42,57,68,76,94,109,120,128,152,162,166,175,176,200,223]).
data(id179,25,552,[5,17,18,22,25,27,32,33,39,59,62,87,91,100,102,111,112,135,137,149,165,168,183,201,204]).
data(id180,25,552,[1,3,4,7,8,9,10,15,18,19,21,41,52,54,73,93,95,123,125,136,138,153,168,261,291]).
data(id181,25,556,[6,8,10,13,19,25,32,37,49,54,58,76,84,91,92,100,107,128,145,156,165,185,195,205,206]).
data(id182,25,556,[3,12,13,15,19,23,27,34,35,39,42,45,48,52,53,87,140,145,158,166,171,184,189,201,227]).
data(id183,25,556,[3,12,13,15,19,23,27,34,35,39,42,45,48,52,53,87,140,145,158,166,171,184,189,201,227]).
data(id184,25,556,[1,5,7,8,9,10,12,14,20,27,31,43,47,50,74,93,97,121,125,139,143,153,167,264,292]).
data(id185,25,562,[2,3,5,8,13,19,20,29,33,47,53,54,64,65,76,93,119,123,142,157,161,180,184,221,259]).
data(id186,25,570,[3,9,10,33,36,38,40,42,50,51,60,69,72,75,77,90,113,140,141,151,152,189,200,229,230]).
data(id187,25,575,[4,6,14,16,31,39,63,69,74,81,88,103,107,111,115,120,131,132,133,147,156,159,164,198,218]).
data(id188,25,576,[1,4,9,11,15,19,22,34,36,53,60,76,82,84,104,126,127,128,153,156,165,174,183,219,237]).
data(id189,25,576,[8,9,10,11,16,30,36,38,45,55,57,65,68,81,84,95,98,100,116,135,144,153,162,279,297]).
data(id190,25,576,[4,6,9,10,17,21,23,25,31,33,36,38,45,50,81,83,115,133,135,144,146,153,162,279,297]).
data(id191,25,580,[2,5,7,10,12,13,19,21,22,29,36,40,61,65,74,101,135,139,161,179,183,192,205,209,236]).
data(id192,25,580,[5,6,11,13,16,17,21,25,34,44,54,68,80,88,100,112,120,135,142,145,170,173,195,215,265]).
data(id193,25,580,[11,12,16,17,29,32,39,41,53,55,59,60,68,70,81,84,92,124,125,128,129,156,171,280,300]).
data(id194,25,593,[13,14,15,35,48,51,55,67,73,79,83,91,94,105,109,116,119,124,133,150,171,173,196,217,226]).
data(id195,25,595,[4,13,18,19,22,35,40,48,58,61,62,77,78,82,83,86,118,149,163,168,187,192,202,206,240]).
data(id196,25,601,[7,8,25,34,41,42,46,48,54,55,62,70,71,74,98,103,116,143,168,169,190,192,193,218,240]).
data(id197,25,603,[7,11,12,14,21,25,32,40,52,56,60,67,68,81,91,92,132,144,149,163,177,191,196,235,263]).
data(id198,25,603,[13,23,26,27,35,44,45,49,53,54,57,66,75,99,101,110,122,126,144,158,175,180,189,234,270]).
data(id199,25,607,[6,8,10,13,19,25,32,37,49,54,58,76,84,91,92,100,107,128,156,185,196,205,206,216,246]).
data(id200,25,609,[9,14,15,17,32,45,47,58,67,74,76,79,80,83,97,111,125,126,150,170,186,188,215,224,235]).
data(id201,25,611,[1,10,22,26,32,41,45,54,57,61,62,66,85,86,87,95,97,101,119,132,136,167,176,268,343]).
data(id202,25,614,[15,22,24,31,33,49,53,54,57,60,63,68,74,81,83,104,109,151,155,163,167,217,229,230,234]).
data(id203,25,634,[15,17,24,26,33,43,44,54,57,60,63,73,79,81,88,109,119,160,161,172,173,227,234,235,239]).
data(id204,25,643,[2,9,21,29,38,40,41,42,58,62,67,76,82,83,85,96,104,166,172,186,192,201,207,250,270]).
data(id205,25,644,[7,9,13,18,19,22,31,49,53,61,66,68,71,87,93,94,119,164,178,192,199,206,227,239,253]).
data(id206,25,655,[10,14,15,21,25,26,31,40,51,53,54,57,65,83,84,86,151,152,173,193,194,215,216,246,288]).
data(id207,25,661,[5,7,17,18,23,31,36,38,41,64,73,77,83,84,102,106,111,161,175,196,203,210,238,248,262]).
```

8.3 Pentominoes

```
:- use_module(library(lists)).
:- use_module(library(ordsets)).
:- use_module(library(clpfd)).
:- use_module(library(structs)).

all :-
size_data(L, D, H),
    bench_all(L, D, H),
    fail.
all.

bench_all(L, D, H) :-
    portray_clause(main(ilex_ff, L, D, H)),
    new(integer, Counter),
    statistics(runtime, _),
    findall(1, terse(ilex_ff, L, D, H, Counter), Ones),
    length(Ones, Sol),
    statistics(runtime, [_,CPU]),
    inc(Counter, BT),
    format('solutions=~d counted backtracks=~d msec=~d\n', [Sol,BT,CPU]),
    dispose(Counter).

run :-
    bench(ilex_ff).

bench(Opt) :-
    size_data(L, D, H),
    bench(Opt, L, D, H),
    fail.
bench(_).

bench(Opt, L, D, H) :-
    portray_clause(main(Opt, L, D, H)),
    new(integer, Counter),
    statistics(runtime, _),
    \+ \+ main(Opt, L, D, H, Counter),
    statistics(runtime, [_,CPU]),
    inc(Counter, BT),
    format('Counted backtracks=~d msec=~d\n', [BT,CPU]),
    dispose(Counter).

size_data(20, 3, 1).
size_data(15, 4, 1).
size_data(12, 5, 1).
size_data(10, 6, 1).
size_data(10, 3, 2).
size_data( 6, 5, 2).
size_data( 5, 4, 3).

main(lex, Length, Depth, Height, Ctr) :-
    problem(Length, Depth, Height, Objects, Shapes, Pieces, _SPieces, Map),
    value_tuples(Length, Depth, Height, Vs),
    lex_labeling(Vs, Pieces, Ctr),
    draw_board(Length, Depth, Height, Objects, Shapes, Map).
main(lex_ff, Length, Depth, Height, Ctr) :-
    problem(Length, Depth, Height, Objects, Shapes, Pieces, _SPieces, Map),
    value_tuples(Length, Depth, Height, Vs),
    lex_ff_labeling(Vs, Pieces, Ctr),
    draw_board(Length, Depth, Height, Objects, Shapes, Map).
main(ilex, Length, Depth, Height, Ctr) :-
    problem(Length, Depth, Height, Objects, Shapes, _Pieces, SPieces, Map),
    N1 is Length*Depth*Height - 1,
    fdset_to_list([[0|N1]], Vs),
    ilex_labeling(Vs, SPieces, Ctr),
    draw_board(Length, Depth, Height, Objects, Shapes, Map).
main(ilex_ff, Length, Depth, Height, Ctr) :-
```

```
    problem(Length, Depth, Height, Objects, Shapes, _Pieces, SPieces, Map),
    N1 is Length*Depth*Height - 1,
    fdset_to_list([[0|N1]], Vs),
    ilex_ff_labeling(Vs, SPieces, Ctr),
    draw_board(Length, Depth, Height, Objects, Shapes, Map).
main(ff, Length, Depth, Height, _Ctr) :- % NO GOOD
    problem(Length, Depth, Height, Objects, Shapes, _Pieces, SPieces, Map),
    append(SPieces, Vars),
    labeling([ff,bisect], Vars),
    draw_board(Length, Depth, Height, Objects, Shapes, Map).
main(min, Length, Depth, Height, _Ctr) :- % NO GOOD
    problem(Length, Depth, Height, Objects, Shapes, _Pieces, SPieces, Map),
    append(SPieces, Vars),
    labeling([min,bisect], Vars),
    draw_board(Length, Depth, Height, Objects, Shapes, Map).

terse(ilex_ff, Length, Depth, Height, Ctr) :-
    problem(Length, Depth, Height, _Objects, _Shapes, _Pieces, SPieces, _Map),
    N1 is Length*Depth*Height - 1,
    fdset_to_list([[0|N1]], Vs),
    ilex_ff_labeling(Vs, SPieces, Ctr).

lex_labeling([], [], _Ctr) :- !.
lex_labeling([V|Vs], Ps1, Ctr) :-
    lex_labeling_fast(Ps1, V, Ps2), !,
    lex_labeling(Vs, Ps2, Ctr).
lex_labeling([V|Vs], Ps1, Ctr) :-
    lex_labeling_assign(V, Ps1, Ps2, Ctr),
    lex_labeling(Vs, Ps2, Ctr).

lex_labeling_fast([[C|P]|Ps1], V, Ps2) :-
    C==V, !,
    cons_nonnil(P, Ps1, Ps2).
lex_labeling_fast([P|Ps1], V, [P|Ps2]) :-
    lex_labeling_fast(Ps1, V, Ps2).

lex_labeling_assign(V, [[C|P]|Ps1], Ps2, Ctr) :-
    ground(C), !,
    Ps2 = [[C|P]|Ps3],
    lex_labeling_assign(V, Ps1, Ps3, Ctr).
lex_labeling_assign(V, [[C|P]|Ps1], Ps3, _Ctr) :-
    assign2(V, C),
    cons_nonnil(P, Ps1, Ps3).
lex_labeling_assign(V, [P|Ps1], [P|Ps2], Ctr) :-
    inc(Ctr, _),
    P = [C|_],
    lex_chain([V,C], [op(#<)]), % CTR: dynlex
    lex_labeling_assign(V, Ps1, Ps2, Ctr).


ilex_labeling([], [], _Ctr) :- !.
ilex_labeling([V|Vs], Ps1, Ctr) :-
    ilex_labeling_fast(Ps1, V, Ps2), !,
    ilex_labeling(Vs, Ps2, Ctr).
ilex_labeling([V|Vs], Ps1, Ctr) :-
    ilex_labeling_assign(V, Ps1, Ps2, Ctr),
    ilex_labeling(Vs, Ps2, Ctr).

ilex_labeling_fast([[C|P]|Ps1], V, Ps2) :-
    C==V, !,
    cons_nonnil(P, Ps1, Ps2).
ilex_labeling_fast([P|Ps1], V, [P|Ps2]) :-
    ilex_labeling_fast(Ps1, V, Ps2).

ilex_labeling_assign(V, [[C|P]|Ps1], Ps2, Ctr) :-
    nonvar(C), !,
    Ps2 = [[C|P]|Ps3],
    ilex_labeling_assign(V, Ps1, Ps3, Ctr).
ilex_labeling_assign(V, [[C|P]|Ps1], Ps3, _Ctr) :-
    C #= V,
    cons_nonnil(P, Ps1, Ps3).
ilex_labeling_assign(V, [P|Ps1], [P|Ps2], Ctr) :-
    inc(Ctr, _),
    P = [C|_],
    C #> V, % CTR: idynlex
    ilex_labeling_assign(V, Ps1, Ps2, Ctr).

ilex_ff_labeling([], [], _Ctr) :- !.
ilex_ff_labeling([V|Vs], Ps1, Ctr) :-
    ilex_ff_labeling_fast(Ps1, V, Ps2), !,
    ilex_ff_labeling(Vs, Ps2, Ctr).
ilex_ff_labeling([V|Vs], Ps1, Ctr) :-
    tag_ipieces(Ps1, Ps2),
    keysort(Ps2, Ps3),
    ilex_ff_labeling_2([V|Vs], Ps3, Ctr).

ilex_ff_labeling_2([], [], _Ctr).
ilex_ff_labeling_2([V|Vs], Ps1, Ctr) :-
    ilex_ff_labeling_assign(V, Ps1, Ps2, Ctr),
    ilex_ff_labeling(Vs, Ps2, Ctr).

ilex_ff_labeling_fast([[C|P]|Ps1], V, Ps2) :-
    C==V, !,
    cons_nonnil(P, Ps1, Ps2).
ilex_ff_labeling_fast([P|Ps1], V, [P|Ps2]) :-
    ilex_ff_labeling_fast(Ps1, V, Ps2).

ilex_ff_labeling_assign(V, [1-P|Ps1], [P|Ps2], Ctr) :- !,
    ilex_ff_labeling_assign(V, Ps1, Ps2, Ctr).
ilex_ff_labeling_assign(V, [_-[C|P]|Ps1], Ps3, _Ctr) :-
    C #= V,
    keys_and_values(Ps1, _, Ps2),
    cons_nonnil(P, Ps2, Ps3).
ilex_ff_labeling_assign(V, [_-P|Ps1], [P|Ps2], Ctr) :-
    inc(Ctr, _),
    P = [C|_],
     C #> V, % CTR: idynlex
    ilex_ff_labeling_assign(V, Ps1, Ps2, Ctr).

lex_ff_labeling([], [], _Ctr) :- !.
lex_ff_labeling([V|Vs], Ps1, Ctr) :-
    lex_ff_labeling_fast(Ps1, V, Ps2), !,
    lex_ff_labeling(Vs, Ps2, Ctr).
lex_ff_labeling([V|Vs], Ps1, Ctr) :-
    tag_pieces(Ps1, Ps2),
    keysort(Ps2, Ps3),
    lex_ff_labeling_2([V|Vs], Ps3, Ctr).

lex_ff_labeling_fast([[C|P]|Ps1], V, Ps2) :-
    C==V, !,
    cons_nonnil(P, Ps1, Ps2).
lex_ff_labeling_fast([P|Ps1], V, [P|Ps2]) :-
    lex_ff_labeling_fast(Ps1, V, Ps2).

lex_ff_labeling_2([], [], _Ctr).
lex_ff_labeling_2([V|Vs], Ps1, Ctr) :-
    lex_ff_labeling_assign(V, Ps1, Ps2, Ctr),
    lex_ff_labeling(Vs, Ps2, Ctr).

lex_ff_labeling_assign(V, [1-[C|P]|Ps1], [[C|P]|Ps2], Ctr) :- !,
    lex_ff_labeling_assign(V, Ps1, Ps2, Ctr).
lex_ff_labeling_assign(V, [_-[C|P]|Ps1], Ps3, _Ctr) :-
    assign2(V, C),
    keys_and_values(Ps1, _, Ps2),
    cons_nonnil(P, Ps2, Ps3).
lex_ff_labeling_assign(V, [_-P|Ps1], [P|Ps2], Ctr) :-
```

```
    inc(Ctr, _),
    P = [C|_],
    lex_chain([V,C], [op(#<)]), % CTR: dynlex
    lex_ff_labeling_assign(V, Ps1, Ps2, Ctr).

inc(Ctr, V) :-
    get_contents(Ctr, contents, V),
    W is V+1,
    put_contents(Ctr, contents, W).

assign2([U,V,W], [X,Y,Z]) :-
    clpfd:'$fd_in_interval'(X, U, U, 1),
    clpfd:'$fd_in_interval'(Y, V, V, 0),
    clpfd:'$fd_in_interval'(Z, W, W, 0),
    clpfd:'$fd_evaluate_indexical'(RC, Global),
    clpfd:evaluate(RC, Global).

cons_nonnil([], Ps, Ps) :- !.
cons_nonnil(X, Ps, [X|Ps]).

tag_pieces([], []).
tag_pieces([P|Ps1], [Tag-P|Ps2]) :-
    P = [[X,Y,Z]|_],
    fd_size(X, S1),
    fd_size(Y, S2),
    fd_size(Z, S3),
    Tag is S1*S2*S3,
    tag_pieces(Ps1, Ps2).

tag_ipieces([], []).
tag_ipieces([P|Ps1], [Tag-P|Ps2]) :-
    P = [X|_],
    fd_size(X, Tag),
    tag_ipieces(Ps1, Ps2).

value_tuples(Length, Depth, Height, Tuples) :-
    LDH is Length*Depth*Height,
    length(Xs, LDH),
    length(Ys, LDH),
    length(Zs, LDH),
    L1 is Length-1,
    D1 is Depth-1,
    H1 is Height-1,
    domain(Xs, 0, L1),
    domain(Ys, 0, D1),
    domain(Zs, 0, H1),
    transpose([Xs,Ys,Zs], Tuples),
    lex_chain(Tuples, [op(#<)]).

draw_board(Length, Depth, Height, Objects, Shapes, Map) :-
    draw_layers(0, Height, Length, Depth, Objects, Shapes, Map).

draw_layers(Z, Z, _, _, _, _, _) :- !.
draw_layers(Z, Height, Length, Depth, Objects, Shapes, Map) :-
    format('+~*c+\n', [Length,0'-]),
    draw_rows(0, Depth, Length, Z, Objects, Shapes, Map),
    format('+~*c+\n', [Length,0'-]),
    nl, nl,
    Z1 is Z+1,
    draw_layers(Z1, Height, Length, Depth, Objects, Shapes, Map).

draw_rows(Y, Y, _, _, _, _, _Map) :- !.
draw_rows(Y, Depth, Length, Z, Objects, Shapes, Map) :-
    write('|'),
    draw_cells(0, Length, Y, Z, Objects, Shapes, Map),
    write('|'),
    nl,
    Y1 is Y+1,
    draw_rows(Y1, Depth, Length, Z, Objects, Shapes, Map).

draw_cells(X, X, _, _, _, _, _Map) :- !.
draw_cells(X, Length, Y, Z, Objects, Shapes, Map) :-
    draw_cell(X, Y, Z, Objects, Shapes, Map),
    X1 is X+1,
    draw_cells(X1, Length, Y, Z, Objects, Shapes, Map).

draw_cell(X2, Y2, Z2, Objects, Shapes, Map) :-
    member(object(OID,SID,[X,Y,Z]), Objects),
    member(shape(SID,[Xt,Yt,Zt],[Xl,Yl,Zl]), Shapes),
    X+Xt =< X2,
    X2 < X+Xt+Xl,
    Y+Yt =< Y2,
    Y2 < Y+Yt+Yl,
    Z+Zt =< Z2,
    Z2 < Z+Zt+Zl, !,
    member(OID-Code, Map),
    write(Code).
draw_cell(_, _, _, _, _, _) :-
    put_code(0' ).

problem(Length, Depth, Height, Objects, Shapes, Pieces, SPieces, Map) :-
    findall(ID-Shape, piece_variant(ID,Shape), Pcs0),
    filter_variants(Pcs0, Pcs1, Length, Depth, Height),
    keyclumped(Pcs1, Pcs2),
    length(Pcs2, NP),
    NSlack is Length*Depth*Height-5*NP,
    NSlack >= 0,
    slack(NSlack, 0, OID0, Slack, []),
    objects(Pcs2, OID0, OID, 1, SID, Map, Objects, Slack),
    shapes(Pcs1, 1, SID, Shapes, []),
    variables(Objects, _Sids, Xs, Ys, Zs),
    L1 is Length-1,
    domain(Xs, 0, L1),
    D1 is Depth-1,
    domain(Ys, 0, D1),
    H1 is Height-1,
    domain(Zs, 0, H1),
    variables(Slack, _, XSs, YSs, ZSs),
    order_slack(XSs, YSs, ZSs),
    OID1 is OID+1,
    OID2 is OID+2,
    SID1 is SID+1,
    SID2 is SID+2,
    L4 is Length+4,
    D4 is Depth+4,
    % CTR: diffn
    diffn([object(OID,SID,[0,0,Height]), object(OID1,SID1,[0,Depth,0]),
           object(OID2,SID2,[Length,0,0])|Objects],
          [shape(0,[0,0,0],[1,1,1]),shape(SID,[0,0,0],[L4,D4,4]),
           shape(SID1,[0,0,0],[L4,4,Height]),shape(SID2,[0,0,0],[4,Depth,Height])|Shapes],
          []),
    object_tuples(Objects, XTuples, YTuples, ZTuples, Cells, []),
    shape_table(Shapes, XTable, YTable, ZTable),
    expand_table(XTable, Length, XTable1, []),
    expand_table(YTable , Depth, YTable1, []),
    expand_table(ZTable, Height, ZTable1, []),
    table(XTuples, XTable1),
    table(YTuples, YTable1),
    table(ZTuples, ZTable1),
    cover(Length, Depth, Height, Cells, Scalars),
    cells_pieces(Objects, Cells, Pieces, Scalars, SPieces).

cover(Length, Depth, Height, Cells, Scalars) :-
    transpose(Cells, [DXs,DYs,DZs]),
    DH1 is Length-1, domain(DXs, 0, DH1),
    LH1 is Depth-1, domain(DYs, 0, LH1),
    LD1 is Height-1, domain(DZs, 0, LD1),
    DH is Depth*Height,
    cells_integers(Cells, DH, Height, Scalars),
    all_distinct(Scalars, [consistency(domain)]), % CTR: alldiff
% packing(DXs, DH),
% LH is Length*Height,
% packing(DYs, LH),
% LD is Length*Depth,
% packing(DZs, LD),
    true.

% packing(Xs, Lim) :-
%     mktasks(Xs, Ts),
%     cumulative(Ts, [limit(Lim),global(true)]).

% mktasks([], []).
% mktasks([O|Os], [task(O,1,E,1,0)|Tasks]) :-
%     E in 0..1000,
%     mktasks(Os, Tasks).

cells_integers([], _, _, []).
cells_integers([C|Cs], DH, H, [I|Is]) :-
    scalar_product([DH,H,1], C, #=, I, [consistency(domain)]), % CTR: scp
    cells_integers(Cs, DH, H, Is).

cells_pieces([], [], [], [], []).
cells_pieces([O|Os], [C|Cs], [[C]|Ps], [I|Is], [[I]|Js]) :-
    O = object(_,SID,_),
    SID==0, !,
    cells_pieces(Os, Cs, Ps, Is, Js).
cells_pieces([_|Os], [C1,C2,C3,C4,C5|Cs], [Piece|Ps], [I1,I2,I3,I4,I5|Is], [SPiece|SPs]) :-
    Piece = [C1,C2,C3,C4,C5],
    SPiece = [I1,I2,I3,I4,I5],
     % lex_chain([C1,C2,C3,C4,C5], [op(#<)]), % CTR: piecelex
    I1#<I2, I2#<I3, I3#<I4, I4#<I5, % CTR: ipiecelex
    cells_pieces(Os, Cs, Ps, Is, SPs).

shape_table(Shapes, XTable, YTable, ZTable) :-
    findall(SID-Coo, shape_coord(Shapes,SID,Coo), L1),
    sort(L1, L2),
    keyclumped(L2, L3),
    split_table(L3, XTable, YTable, ZTable).

expand_table([], _) --> [].
expand_table([Row|Table], N) -->
    expand_row(0, N, Row),
    expand_table(Table, N).

expand_row(N, N, _) --> !.
expand_row(I, N, Row) --> [[S,I,I1,I2,I3,I4,I5]],
    {Row = [S,O1,O2,O3,O4,O5]},
    {I1 is I+O1},
    {I2 is I+O2},
    {I3 is I+O3},
    {I4 is I+O4},
    {I5 is I+O5},
    {J is I+1},
    expand_row(J, N, Row).

split_table([], [], [], []).
split_table([SID-Coos|L1], [Xrow|Xs], [Yrow|Ys], [Zrow|Zs]) :-
    transpose([[SID,SID,SID]|Coos], [Xrow,Yrow,Zrow]),
    split_table(L1, Xs, Ys, Zs).

shape_coord(Shapes, SID, [X,Y,Z]) :-
    member(shape(SID,[T1,T2,T3],[L1,L2,L3]), Shapes),
    E1 is T1+L1-1,
    E2 is T2+L2-1,
    E3 is T3+L3-1,
    X in T1..E1,
    Y in T2..E2,
    Z in T3..E3,
    indomain(X),
    indomain(Y),
    indomain(Z).

object_tuples([], [], [], []) --> [].
object_tuples([object(_,SID,Orig)|Objects], XTuples, YTuples, ZTuples) --> [Orig],
    {SID==0}, !,
    object_tuples(Objects, XTuples, YTuples, ZTuples).
object_tuples([object(_,SID,[X,Y,Z])|Objects], [XT|XTuples], [YT|YTuples], [ZT|ZTuples]) --> [T1,T2,T3,T4,T5],
    {XT = [SID,X,X1,X2,X3,X4,X5]},
    {YT = [SID,Y,Y1,Y2,Y3,Y4,Y5]},
    {ZT = [SID,Z,Z1,Z2,Z3,Z4,Z5]},
    {T1=[X1,Y1,Z1]},
    {T2=[X2,Y2,Z2]},
    {T3=[X3,Y3,Z3]},
    {T4=[X4,Y4,Z4]},
    {T5=[X5,Y5,Z5]},
    object_tuples(Objects, XTuples, YTuples, ZTuples).

order_slack([], [], []) :- !.
order_slack(Xs, Ys, Zs) :-
    transpose([Xs,Ys,Zs], Tuples),
    lex_chain(Tuples, [op(#<)]).

slack(0, OID, OID) --> !.
slack(J, OID0, OID) --> [object(OID0,0,[_,_,_])],
    {I is J-1},
    {OID1 is OID0+1},
    slack(I, OID1, OID).

objects([], OID, OID, SID, SID, []) --> [].
objects([Tag-Clump|Pcs], OID0, OID, SID0, SID, [OID0-Tag|Map]) --> [object(OID0,SID3,[_,_,_])],
    {OID1 is OID0+1},
    {length(Clump, N)},
    {SID1 is SID0+N},
    {SID2 is SID1-1},
    {SID3 in SID0..SID2},
    objects(Pcs, OID1, OID, SID1, SID, Map).

shapes([], SID, SID) --> [].
shapes([_-Parts|Pcs], SID0, SID) -->
    parts(Parts, SID0),
    {SID1 is SID0+1},
    shapes(Pcs, SID1, SID).

parts([], _) --> [].
parts([Tran-Len|Ps], SID) --> [shape(SID,Tran,Len)],
    parts(Ps, SID).

variables([], [], [], [], []).
variables([object(_,S,[X,Y,Z])|Objects], [S|Ss], [X|Xs], [Y|Ys], [Z|Zs]) :-
    variables(Objects, Ss, Xs, Ys, Zs).

filter_variants([], [], _, _, _).
filter_variants([T-S|Ss1], [T-S|Ss2], L, D, H) :-
    variant_fits(S, L, D, H), !,
    filter_variants(Ss1, Ss2, L, D, H).
filter_variants([_|Ss1], Ss2, L, D, H) :-
    filter_variants(Ss1, Ss2, L, D, H).

variant_fits([], _, _, _).
variant_fits([[T1,T2,T3]-[L1,L2,L3]|S], L, D, H) :-
    T1+L1 =< L,
    T2+L2 =< D,
    T3+L3 =< H,
    variant_fits(S, L, D, H).

piece_variant(ID, Variant) :-
    piece(ID, Shape1),
    sort(Shape1, Shape2),
    piece_closure([Shape2], [Shape2], Closure),
    member(Variant, Closure).

% piece_closure(New, Sofar, Closure).
piece_closure([], Shapes, Shapes).
piece_closure([B1|New1], Sofar1, Closure) :-
    findall(B2, transform_shape(B1,B2), Shapes2),
    sort(Shapes2, Shapes3),
    ord_union(Sofar1, Shapes3, Sofar2, New2),
    append(New1, New2, New3),
    piece_closure(New3, Sofar2, Closure).

transform_shape(S1, S4) :-
    axis(Axis),
    rotate(S1, S2, Axis),
    min3(S2, Min),
    translate(S2, Min, S3),
    sort(S3, S4).

axis(x). axis(y). axis(z).

rotate([], [], _).
rotate([Box1|Shape1], [Box2|Shape2], Axis) :-
    rotate_box(Axis, Box1, Box2),
    rotate(Shape1, Shape2, Axis).

rotate_box(x, [T1,T2,T3]-[L1,L2,L3], [T1,T3,NT2]-[L1,L3,L2]) :-
    rot_negate(T2, L2, NT2).
rotate_box(y, [T1,T2,T3]-[L1,L2,L3], [NT3,T2,T1]-[L3,L2,L1]) :-
    rot_negate(T3, L3, NT3).
rotate_box(z, [T1,T2,T3]-[L1,L2,L3], [T2,NT1,T3]-[L2,L1,L3]) :-
    rot_negate(T1, L1, NT1).

rot_negate(T1, L, T2) :-
    T2 is (1-L)-T1.

min3([Orig-_|Shape], Min) :-
    min3(Shape, Orig, Min).

min3([], Min, Min).
min3([[X1,Y1,Z1]-_|Shape], [X2,Y2,Z2], Min) :-
    X3 is min(X1,X2),
    Y3 is min(Y1,Y2),
    Z3 is min(Z1,Z2),
    min3(Shape, [X3,Y3,Z3], Min).

translate([], _, []).
translate([[X1,Y1,Z1]-Offset|Shape1], Min, [[X3,Y3,Z3]-Offset|Shape3]) :-
    Min = [X2,Y2,Z2],
    X3 is -(X1,X2),
    Y3 is -(Y1,Y2),
    Z3 is -(Z1,Z2),
    translate(Shape1, Min, Shape3).

:- dynamic piece/2.

% piece(ID, list of Part).
% Part --> Orig-Length

% Overlapping variant

piece(i, [[0,0,0]-[5,1,1]]).
piece(y, [[1,0,0]-[1,4,1],[0,2,0]-[2,1,1]]).
piece(l, [[0,0,0]-[1,4,1],[0,0,0]-[2,1,1]]).
piece(n, [[0,0,0]-[1,3,1],[1,2,0]-[1,2,1]]).
piece(v, [[0,0,0]-[3,1,1],[0,0,0]-[1,3,1]]).
piece(t, [[0,2,0]-[3,1,1],[1,0,0]-[1,3,1]]).
piece(w, [[0,0,0]-[2,1,1],[1,0,0]-[1,2,1],[1,1,0]-[2,1,1],[2,1,0]-[1,2,1]]).
piece(x, [[0,1,0]-[3,1,1],[1,0,0]-[1,3,1]]).
piece(u, [[0,0,0]-[3,1,1],[0,0,0]-[1,2,1],[2,0,0]-[1,2,1]]).
piece(z, [[1,0,0]-[1,3,1],[1,0,0]-[2,1,1],[0,2,0]-[2,1,1]]).
piece(f, [[1,0,0]-[1,3,1],[0,1,0]-[2,1,1],[1,2,0]-[2,1,1]]).
piece(p, [[0,0,0]-[1,3,1],[1,1,0]-[1,2,1]]).
```

8.4 Orthogonal Packing (Clautiaux Instances)

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).

runall :-
    runall(primal([bisect])).

runall(Srch) :-
    retractall(btr_time/2),
    instance(ID, _, _),
    order(Order),
    statistics(runtime, _),
    countall(ID, Srch, N, Order),
    statistics(runtime, [_,Time]),
    fd_statistics(backtracks, Btr),
    logit('instance=~w order=~w solutions=~d backtracks=~d time=~d\n', [ID,Order,N,Btr,Time]),
    assertz(btr_time(Btr,Time)),
    fail.
runall(_) :-
    findall(B-T, btr_time(B,T), Btr1),
    keysort(Btr1, Btr2),
    findall(T-B, btr_time(B,T), Time1),
    keysort(Time1, Time2),
    tell('squares.dat'),
    write_plot_data(Btr2, Time2, 0),
    told,
    keys_and_values(Time2, Times, Btrs),
    sumlist(Btrs, SumBtr),
    sumlist(Times, SumTime),
    format('TOTAL backtracks=~d time=~d\n', [SumBtr,SumTime]).

runfirst :-
    runfirst(primal([bisect])).

runfirst(Srch) :-
    retractall(btr_time/2),
    instance(ID, _, _),
    order(Order),
    statistics(runtime, _),
    countfirst(ID, Srch, N, Order),
    statistics(runtime, [_,Time]),
    fd_statistics(backtracks, Btr),
    logit('instance=~w order=~w solutions=~d backtracks=~d time=~d\n', [ID,Order,N,Btr,Time]),
    assertz(btr_time(Btr,Time)),
    fail.
runfirst(_) :-
    findall(B-T, btr_time(B,T), Btr1),
    keysort(Btr1, Btr2),
    findall(T-B, btr_time(B,T), Time1),
    keysort(Time1, Time2),
    tell('squares.dat'),
    write_plot_data(Btr2, Time2, 0),
    told,
    keys_and_values(Time2, Times, Btrs),
    sumlist(Btrs, SumBtr),
    sumlist(Times, SumTime),
    format('TOTAL backtracks=~d time=~d\n', [SumBtr,SumTime]).

write_plot_data([], [], _).
write_plot_data([B-_|Bs], [T-_|Ts], I) :-
    J is I+1,
    format('~d ~d ~d\n', [J,B,T]),
    write_plot_data(Bs, Ts, J).

logit(Fmt, Args) :-
    prolog_flag(argv, [Logfile]), !,
    open(Logfile, append, S),
    format(S, Fmt, Args),
    close(S).
logit(Fmt, Args) :-
    format(Fmt, Args).

countall(ID, Search, N, Order) :-
    findall(1, solve(ID,Search,_,_,Order), L),
    length(L, N).

countfirst(ID, Search, N, Order) :-
    findall(1, (solve(ID,Search,_,_,Order)->true), L),
    length(L, N).

solve(ID, Search, Xs, Ys, Order) :-
    constraints(ID, Xs, Ys),
    sum_dom_size(Xs, 0, T1),
    sum_dom_size(Ys, T1, Total),
    logit('instance=~w order=~w total domain size=~d\n', [ID,Order,Total]),
    search(Search, Xs, Ys, Order).

sum_dom_size([], T, T).
sum_dom_size([X|Xs], T0, T) :-
    fd_size(X, S),
    T1 is T0+S,
    sum_dom_size(Xs, T1, T).

search(primal(OPT), Xs, Ys, Order) :-
    sort2(Order, Xs, Ys, Xs1, Ys1),
    labeling(OPT, Xs1),
    labeling(OPT, Ys1).

sort2(xy, X, Y, X, Y).
sort2(yx, X, Y, Y, X).

constraints(ID, Xs, Ys) :-
    Bin = bin(_Length,_Height),
    instance(ID, Bin, Items),
    sort(Items, Shapes1),
    items_objects(Items, Bin, Shapes1, _XTasks, _YTasks, Xs, Ys, 0, Objects, []),
    % shape_orig_pairs(Objects, L1),
    % keyclumped(L1, L2),
    % lex_chain_each(L2),
    items_shapes(Shapes1, Shapes2, 0),
    diffn(Objects, Shapes2, []),
    true.

shape_orig_pairs([], []).
shape_orig_pairs([object(_,S,O)|Objects], [S-O|SOs]) :-
    shape_orig_pairs(Objects, SOs).

lex_chain_each([]).
lex_chain_each([_-Origs|L]) :-
    lex_chain(Origs, [op(#<)]),
    lex_chain_each(L).

items_objects([], _, _, [], [], [], [], _) --> [].
items_objects([item(L,H)|Items], bin(Len,Height), Shapes,
    [task(X,L,XE,H,0)|XTasks], [task(Y,H,YE,L,0)|YTasks], [X|Xs], [Y|Ys], O) --> [object(O,S,[X,Y])],
    {domain([XE,YE], 0, 100)},
    {nth0(S, Shapes, item(L,H))},
    {UBX is Len-L},
    {UBY is Height-H},
    {X in 0..UBX},
    {Y in 0..UBY},
    {O1 is O+1},
    items_objects(Items, bin(Len,Height), Shapes, XTasks, YTasks, Xs, Ys, O1).

items_shapes([], [], _).
items_shapes([item(L,H)|Items], [shape(S,[0,0],[L,H])|Shapes], S) :-
    S1 is S+1,
    items_shapes(Items, Shapes, S1).

order(xy).
order(yx).

:- dynamic instance/3.
instance(e00X23,bin(20,20),
[item(6,11),item(4,15),item(15,4),item(17,2),item(11,3),item(9,3),item(9,3),item(5,5),item(5,2),
 item(1,7), item(1,7),item(1,7),item(1,5),item(1,5),item(1,4),item(1,4),item(1,4),item(1,3),
 item(1,3),item(1,3),item(1,2),item(1,2),item(1,2)]).

instance(e00N23,bin(20,20),
[item(20,3),item(4,13),item(7,7),item(11,4),item(7,5),item(5,7),item(16,2),item(5,5),item(1,8),
 item(1,6), item(1,6),item(1,5),item(1,5),item(1,5),item(1,5),item(2,2),item(1,4),item(2,2),
 item(2,2),item(1,3),item(1,3),item(1,3),item(1,3)]).

instance(e05N15,bin(20,20),
[item(3,17),item(5,10),item(5,8),item(5,8),item(3,13),item(6,5),item(2,14),item(11,2),item(2,11),
 item(1,16),item(1,16),item(1,13),item(3,2),item(1,5),item(1,2)]).

instance(e05X15,bin(20,20),
[item(11,6),item(5,13),item(9,6),item(11,4),item(7,6),item(3,13),item(2,10),item(1,13),item(2,5),
 item(4,2),item(1,7),item(1,4),item(1,3),item(1,3),item(1,2)]).

instance(e05F20,bin(20,20),
[item(6,10),item(19,3),item(11,4),item(8,5),item(2,17),item(5,6),item(3,9),item(5,3),item(4,3),
 item(1,11),item(3,3),item(1,7),item(1,7),item(1,7),item(1,5),item(1,5),item(1,3),item(1,3),
 item(1,2),item(1,2)]).

instance(e04F20,bin(20,20),
[item(9,7),item(4,13),item(7,7),item(17,2),item(3,11),item(6,5),item(1,17),item(8,2),item(1,13),
 item(1,13),item(1,10),item(1,10),item(1,9),item(3,3),item(2,4),item(1,5),item(2,2),item(1,3),
 item(1,3),item(1,3)]).

instance(e13X15,bin(20,20),
[item(9,7),item(9,7),item(9,7),item(5,7),item(2,17),item(12,2),item(1,15),item(1,13),item(2,5),
 item(1,9),item(1,6),item(1,5),item(2,2),item(1,2),item(1,2)]).

instance(e10N15,bin(20,20),
[item(11,6),item(10,5),item(4,11),item(2,17),item(2,14),item(2,11),item(1,20),item(1,17),item(7,2),
 item(1,13),item(1,13),item(1,13),item(1,11),item(1,9),item(2,3)]).

instance(e03N16,bin(20,20),
[item(13,5),item(16,4),item(4,11),item(15,2),item(2,13),item(5,5),item(2,11),item(10,2),item(6,3),
 item(1,17),item(5,3),item(1,11),item(1,9),item(1,9),item(1,9),item(1,4)]).

instance(e20F15,bin(20,20),
[item(13,5),item(13,4),item(5,10),item(17,2),item(3,9),item(2,11),item(8,2),item(6,2),item(1,12),
 item(3,3),item(3,2),item(1,5),item(1,4),item(1,3),item(1,3)]).

instance(e04N15,bin(20,20),
[item(19,3),item(5,11),item(10,5),item(15,3),item(2,16),item(9,3),item(3,9),item(1,17),item(8,2),
 item(1,13),item(1,13),item(1,12),item(1,11),item(1,5),item(1,4)]).

instance(e03N15,bin(20,20),
[item(6,11),item(13,4),item(8,6),item(5,9),item(3,13),item(16,2),item(3,9),item(2,11),item(1,17),
 item(1,10),item(1,9),item(3,3),item(1,5),item(1,4),item(1,3)]).

instance(e10X15,bin(20,20),
[item(8,8),item(9,7),item(14,3),item(17,2),item(16,2),item(2,13),item(2,12),item(1,17),item(1,13),
 item(1,11),item(1,9),item(2,4),item(2,4),item(1,5),item(2,2)]).

instance(e07X15,bin(20,20),
[item(7,9),item(12,5),item(5,9),item(11,4),item(11,4),item(4,11),item(1,17),item(1,13),item(4,3),
 item(3,4),item(3,2),item(2,2),item(1,3),item(1,3),item(1,2)]).
```

```
instance(e05N17,bin(20,20),
[item(13,5),item(19,3),item(10,5),item(13,3),item(3,13),item(5,6),item(4,5),item(1,20),item(1,13),
 item(1,13),item(1,13),item(3,2),item(1,4),item(2,2),item(1,3),item(1,3),item(1,2)]).

instance(e08N15,bin(20,20),
[item(13,4),item(4,13),item(3,17),item(7,7),item(7,7),item(16,3),item(1,20),item(1,10),item(3,3),
 item(1,8),item(1,5),item(1,5),item(1,4),item(1,4),item(1,3)]).

instance(e07F15,bin(20,20),
[item(13,5),item(12,4),item(11,4),item(2,20),item(2,17),item(8,4),item(3,10),item(2,10),item(2,8),
 item(1,15),item(4,3),item(1,7),item(1,5),item(1,3),item(1,3)]).

instance(e03F18,bin(20,20),
[item(4,16),item(7,8),item(18,3),item(2,17),item(2,16),item(13,2),item(5,5),item(7,3),item(10,2),
 item(1,12),item(3,3),item(3,3),item(1,6),item(1,5),item(1,5),item(2,2),item(1,4),item(1,2)]).

instance(e04N18,bin(20,20),
[item(11,6),item(17,3),item(5,10),item(12,4),item(17,2),item(2,15),item(7,3),item(1,20),item(1,20),
 item(1,14),item(2,4),item(1,4),item(1,4),item(1,4),item(1,3),item(1,3),item(1,3),item(1,2)]).

instance(e02F20,bin(20,20),
[item(7,8),item(3,15),item(4,11),item(3,14),item(8,4),item(13,2),item(6,4),item(2,11),item(6,3),
 item(1,15),item(6,2),item(1,11),item(3,3),item(4,2),item(2,4),item(1,7),item(2,3),item(1,3),
 item(1,2),item(1,2)]).

instance(e04F17,bin(20,20),
[item(9,7),item(7,9),item(10,4),item(19,2),item(2,16),item(3,10),item(8,3),item(11,2),item(3,6),
 item(1,13),item(2,6),item(3,3),item(1,7),item(2,2),item(2,2),item(1,3),item(1,2)]).

instance(e13N10,bin(20,20),
[item(8,8),item(5,12),item(17,3),item(4,12),item(3,11),item(11,3),item(10,3),item(2,12),item(1,3),
 item(1,2)]).

instance(e04F15,bin(20,20),
[item(5,13),item(7,8),item(13,4),item(10,4),item(3,13),item(9,3),item(3,9),item(13,2),item(7,2),
 item(7,2),item(1,9),item(1,5),item(2,2),item(1,3),item(1,3)]).

instance(e03N17,bin(20,20),
[item(5,13),item(16,4),item(10,5),item(7,7),item(17,2),item(2,16),item(6,4),item(1,20),item(5,3),
 item(2,5),item(1,5),item(1,5),item(1,4),item(1,3),item(1,3),item(1,3),item(1,3)]).

instance(e00N15,bin(20,20),
[item(13,5),item(16,4),item(4,14),item(10,4),item(11,3),item(7,4),item(11,2),item(1,17),item(8,2),
 item(5,3),item(1,13),item(4,3),item(2,6),item(1,5),item(1,2)]).

instance(e20X15,bin(20,20),
[item(4,16),item(17,3),item(7,7),item(2,17),item(14,2),item(4,6),item(1,20),item(8,2),item(4,3),
 item(3,2),item(1,5),item(2,2),item(1,3),item(1,3),item(1,2)]).

instance(e05F15,bin(20,20),
[item(13,5),item(11,5),item(10,5),item(6,8),item(19,2),item(7,5),item(15,2),item(2,9),item(1,11),
 item(1,9),item(4,2),item(1,6),item(1,3),item(1,2),item(1,2)]).

instance(e02F17,bin(20,20),
[item(4,14),item(13,4),item(6,7),item(5,8),item(15,2),item(15,2),item(2,14),item(7,3),item(3,7),
 item(1,20),item(2,5),item(5,2),item(3,3),item(1,9),item(2,3),item(1,6),item(1,3)]).

instance(e02F22,bin(20,20),
[item(3,18),item(10,4),item(3,13),item(13,3),item(19,2),item(5,7),item(5,6),item(3,9),item(3,5),
 item(1,12),item(5,2),item(1,9),item(2,4),item(1,6),item(1,5),item(1,5),item(2,2),item(2,2),
 item(2,2),item(2,2),item(1,2),item(1,2)]).

instance(e04F19,bin(20,20),
[item(9,7),item(6,9),item(8,5),item(9,4),item(3,11),item(3,10),item(5,5),item(3,7),item(2,9),
 item(1,13),item(5,2),item(1,9),item(4,2),item(1,7),item(1,6),item(1,4),item(1,3),item(1,2),
 item(1,2)]).

instance(e05F18,bin(20,20),
[item(5,11),item(5,11),item(6,9),item(2,16),item(2,16),item(14,2),item(12,2),item(6,4),item(8,2),
 item(1,11),item(5,2),item(3,3),item(1,8),item(1,7),item(1,7),item(1,3),item(1,3),item(1,2)]).

instance(e08F15,bin(20,20),
[item(8,8),item(7,8),item(4,13),item(9,5),item(7,5),item(2,15),item(1,17),item(4,4),item(1,12),
 item(5,2),item(1,9),item(1,9),item(2,3),item(2,2),item(1,3)]).

instance(e04N17,bin(20,20),
[item(3,20),item(11,5),item(3,17),item(5,10),item(9,4),item(8,4),item(2,14),item(1,20),item(1,14),
 item(1,12),item(2,5),item(3,2),item(1,4),item(1,3),item(1,3),item(1,2),item(1,2)]).

instance(e13N15,bin(20,20),
[item(9,7),item(8,5),item(19,2),item(19,2),item(17,2),item(2,16),item(14,2),item(13,2),item(1,20),
 item(1,11),item(2,3),item(1,4),item(1,4),item(1,3),item(1,2)]).

instance(e15N15,bin(20,20),
[item(13,5),item(14,4),item(4,11),item(2,20),item(15,2),item(1,20),item(2,7),item(1,14),item(1,13),
 item(1,13),item(1,13),item(1,11),item(1,5),item(1,3),item(1,2)]).

instance(e00N10,bin(20,20),
[item(17,6),item(4,14),item(7,8),item(7,7),item(5,7),item(8,4),item(2,13),item(4,5),item(1,13),
 item(1,11)]).

instance(e02N20,bin(20,20),
[item(9,7),item(7,9),item(3,20),item(12,3),item(10,3),item(5,5),item(4,5),item(1,20),item(1,18),
 item(1,13),item(5,2),item(4,2),item(1,6),item(1,6),item(1,5),item(2,2),item(1,2),item(1,2),
 item(1,2),item(1,2)]).

instance(e03N10,bin(20,20),
[item(14,12),item(19,3),item(6,9),item(16,2),item(14,2),item(4,4),item(1,11),item(1,9),item(1,7),
 item(2,3)]).

instance(e07N10,bin(20,20),
[item(11,9),item(5,13),item(11,5),item(7,7),item(3,14),item(12,2),item(1,13),item(1,10),item(1,9),
 item(1,6)]).

instance(e07N15,bin(20,20),
[item(11,6),item(13,4),item(3,17),item(4,11),item(2,20),item(5,7),item(1,15),item(1,13),item(1,13),
 item(1,11),item(1,9),item(3,3),item(1,8),item(1,7),item(1,3)]).

instance(e10N10,bin(20,20),
[item(15,9),item(6,11),item(13,5),item(4,7),item(5,3),item(1,14),item(1,13),item(1,11),item(1,8),
 item(1,5)]).
```

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. N. Beldiceanu, Q. Guo, and S. Thiel. Non-overlapping constraints between convex polytopes. In T. Walsh, editor, *Proc. CP'2001*, volume 2239 of *LNCS*, pages 392–407. Springer-Verlag, 2001.
3. N. Beldiceanu and M. Carlsson. Sweep as a generic pruning technique applied to the non-overlapping rectangles constraints. In T. Walsh, editor, *Proc. CP'2001*, volume 2239 of *LNCS*, pages 377–391. Springer-Verlag, 2001.
4. F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers and Operation Research*, to appear.
5. 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.
6. A. Colmerauer and B. Gilleta. Solving the three-dimensional pentamino puzzle. Technical report, Laboratoire d'Informatique de Marseille, 1999. http://www.lim.univ-mrs.fr/ colmer/ArchivesPublications/Giletta/misc99.pdf.
7. 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.
8. F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, to appear.
9. Pascal Van Hentenryck, Vijay Saraswat, and Yves Deville. Constraint processing in cc(FD). Manuscript, 1991.
10. Gregory Sidebottom. *A Language for Optimizing Constraint Propagation*. PhD thesis, Simon Fraser University, 1993.
11. Alessandro Dal Pal‘u, Agostino Dovier, and Enrico Pontelli. A new constraint solver for 3D lattices and its application to the protein folding problem. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005*, volume 3835 of *LNCS*, pages 48–63. Springer-Verlag, 2005.