Alcom-FT Technical Report Series
ALCOMFT-TR-02-47

# Non-overlapping Constraints between Convex Polytopes*

Nicolas Beldiceanu[1], Qi Guo[2**], and Sven Thiel[3]

[1]SICS, Lägerhyddsvägen 18, SE-75237 Uppsala, Sweden
nicolas@sics.se
[2]Department of Mathematics, Harbin Institute of Technology, 150006 Harbin, China
guo@math.uu.se
[3]MPI für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
sthiel@mpi-sb.mpg.de

**Abstract.** This paper deals with *non-overlapping* constraints between convex polytopes. Non-overlapping detection between fixed objects is a fundamental geometric primitive that arises in many applications. However from a constraint perspective it is natural to extend the previous problem to a non-overlapping constraint between two objects for which both positions are not yet fixed. A first contribution is to present theorems for convex polytopes which allow coming up with general necessary conditions for non-overlapping. These theorems can be seen as a generalization of the notion of *compulsory part* which was introduced in 1984 by Lahrichi and Gondran [7] for managing non-overlapping constraint between rectangles. Finally, a second contribution is to derive from the previous theorems efficient filtering algorithms for two special cases: the non-overlapping constraint between two convex polygons as well as the non-overlapping constraint between *d*-dimensional boxes.

## 1 Introduction

The first part of this paper introduces necessary conditions for the *non-overlapping* constraint between convex polytopes. A convex polytope[1] [4] is defined by the convex hull of a finite number of points. Non-overlapping detection between fixed objects is a fundamental geometric primitive that arises in many applications. However from a constraint perspective it is natural to extend the previous problem to a non-overlapping constraint between objects for which the positions are not yet fixed. Concretely this means that we first want to detect as soon as possible before fixing completely two polytopes whether they will overlap or not. Secondly, we would like to find out the portion of space where placing a polytope will necessarily cause it to overlap with another not yet completely fixed polytope. For instance consider the illustrative example given in Fig. 1. We have a rectangle $R_1$ of length 3 and height 1 which must be included within box $B$ and which should not overlap the rectangle $R_2$ of length 2 and height 4. We want

* Partly supported by the IST Program of the EU under contract number IST-1999-14186, (ALCOM-FT).
** Currently at: Department of Mathematics, Uppsala University, SE-75237 Uppsala, Sweden.
1 From now on, the term *polytope* will refer to a convex polytope.

to find out that the origin of $R_2$ (i.e. the leftmost lower corner of $R_2$) can't be put within box $F$.


(A) Rectangles to place

(B) Domain of $R_1$ and forbidden domain $F$ for the origin of $R_2$

**Fig. 1.** An illustrative example of a forbidden domain

Within constraint programming [10], elaborate shapes are currently approximated [6] by a set of rectangles, where the origin of each rectangle is linked to the origin of another rectangle by an external equality constraint. Since a huge number of rectangles may be required in order to approximate a specific shape, this increases the problem's size. This also leads to poor constraint propagation since each small rectangle is considered separately from the other rectangles to which it is linked by an external equality constraint. Practical motivations for having more than two dimensions are as follows. First it allows modelling both the geographical location and the time location of objects that should not be at the same location at the same instant. Secondly introducing a third dimension for a two-dimensional placement problem is also valuable for including relaxation directly within the constraint: in this third dimension, the coordinates of the origin of the objects which could be placed (respectively not placed) will be set to 0 (respectively to a value greater than 0).

The second part of this paper presents efficient filtering algorithms for two special cases of the non-overlapping constraints: the non-overlapping constraint between 2 convex polygons as well as the non-overlapping constraint between 2 *d*-dimensional boxes.

The next section introduces gradually the different definitions needed for describing the objects we consider, as well as the notion of *intersection* between these objects. Sect. 3 defines the concept of *overlapping polytope*, which is a portion of space where placing the origin of one polytope will lead it to overlap with another not yet fixed polytope. This extends the concept of *compulsory part* (i.e. the intersection of all the feasible instances of an object to place) which was presented for the case of rectangles in [7]. Finally based on the theorems of Sect. 3, we respectively derive in Sect. 4 and 5 two efficient filtering algorithms for the case of convex polygons and for the case of *d*-dimensional boxes.

## 2 Background, Definitions and Notation

The purpose of this section is twofold. First, it describes the objects we consider for our placement problem. Second, it introduces the notion of intersection between these objects.

**Definition 1** *domain variable*

A *domain variable* is a variable that ranges over a finite set of integers; $\underline{V}$ and $\overline{V}$ respectively denote the minimum and maximum values of variable $V$.

**Definition 2** *fixed polytope*

A *fixed polytope* in $IR^d$ is a polytope defined by *k* vertices and their respective integer coordinates, such that all points of the polytope belong to the convex hull of the *k* vertices.

**Definition 3** *shape polytope*

A *shape polytope* in $IR^d$ is a polytope defined by its *k* vertices and their respective integer coordinates, such that all points of the polytope belong to the convex hull of the *k* vertices, and such that one of its vertices has only zero coordinates. This specific vertex is called the *origin* of the shape polytope.

The shape polytope describes the shape of the objects we have to place, while a fixed polytope gives the possible positions for the origin of a shape.


(A) Fixed polytope (B) Shape polytope

**Fig. 2.** Examples of polytopes

Part (A) of Fig. 2 gives an example of a fixed polytope, while part (B) describes a shape polytope. The next four definitions are introduced in order to define the notion of *intersection* between two fixed polytopes.

**Definition 4** *interior point*

A point $X$ of a fixed polytope $P$ is called an *interior point* if there is an $r > 0$ such that $Ball(X,r) \subset P$, where $Ball(X,r) = \{Y : dist(Y,X) < r\}$, and $dist(Y,X)$ is the Euclidean distance between points $X$ and $Y$.

**Definition 5** *k-dimensional hyperplane*

$H \subset IR^d$ is called a *k-dimensional hyperplane* if $H = x + R_k$, where $x \in IR^d$ is a fixed point and $R_k$ is a *k*-dimensional subspace of $IR^d$.

**Definition 6** *dimension of a fixed polytope*

If there is a $k$-dimensional hyperplane that contains a fixed polytope $P$ and no any $k-1$-dimensional hyperplane contains $P$ then $k$ is called the *dimension* of $P$.

**Definition 7** *relative interior point*

Let $P$ be a fixed polytope of dimension $k$. Then there exists a $k$-dimensional hyperplane $H$ such that $P \subseteq H$. If a point $X$ of $P$ is an interior point of $P$ considered only in $H$, then $X$ is called a *relative interior point* of $P$.

In order to illustrate the previous definitions let us consider a fixed polytope $P$ of $IR^2$ that corresponds to a line-segment between points $X1$ and $X2$. $P$ has no interior points, but the dimension of $P$ is $1$ and all points of $P$ that are distinct from $X1$ and $X2$ are relative interior points of $P$.

**Definition 8** *intersection of fixed polytopes*

Two fixed polytopes $P$ and $Q$ *intersect* (i.e. overlap) if $P$ and $Q$ have a common relative interior point.

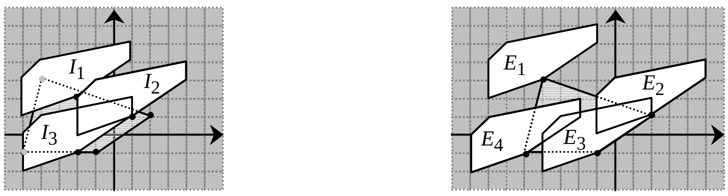
Part (A) of Fig. 3 gives three pairs $(P_1, P_2)$, $(P_3, P_4)$ and $(P_5, P_6)$ of intersecting polytopes, while part (B) shows seven pairwise non-intersecting polytopes. Note that, according to Definition 8, point $P_{13}$ does not overlap rectangle $P_9$ since $P_{13}$ has no relative interior points.


(A) Intersecting polytopes (B) Non-intersecting polytopes

**Fig. 3.** Illustration of the definition of intersection

Throughout the paper we use the following notations:

– $|P|$ designates the *number of vertices* of a fixed or of a shape polytope $P$,
– $\min_i(P)$ (respectively $\max_i(P)$) is the *minimum* (respectively *maximum*) *value* of the coordinates on the $i$ axis of the vertices of a fixed polytope $P$,
– $P^\bullet$ designates the set of *relative interior points* of $P$,
– $bd(P)$ denotes the set of points of $P$ which do not belong to $P^\bullet$ (i.e. the boundary of $P$),
– $conv(X_1, X_2, \ldots, X_n)$ denotes the *convex hull* of a given set of points $X_1, X_2, \ldots, X_n$,
– Finally, $box(O_1, O_2, \ldots, O_d)$, where $O_1, O_2, \ldots, O_d$ are domain variables, is the fixed polytope defined as the points of $[\underline{O_1}, \overline{O_1}] \times [\underline{O_2}, \overline{O_2}] \times \ldots \times [\underline{O_d}, \overline{O_d}]$.




(A) Three instances of a family (B) Extremum polytopes of a family

**Fig. 4.** A family of polytopes

**Definition 9** *family of polytopes*

A *family* $F$ *of polytopes* in $IR^d$ is a set of fixed polytopes defined by:

– A shape polytope $P_{shape}(F)$ in $IR^d$ that describes the shape of the polytopes of the family,
– A fixed polytope $P_{origin}(F)$ in $IR^d$ that gives the initial possible placements for the origin of the polytope $P_{shape}(F)$,

– A tuple $\langle O_1, O_2, \ldots, O_d \rangle$ of $d$ domain variables that further restricts the possible placements for the origin of the polytope $P_{shape}(F)$ to the polytope $P_o(F)$ defined by $box(O_1, O_2, \ldots, O_d)$.

The members of $F$ are fixed polytopes that are obtained by fixing the origin of $P_{shape}(F)$ to any integer point that is not located outside $P_{origin}(F) \cap P_o(F)$. The tuple $\langle O_1, O_2, \ldots, O_d \rangle$ is called the *origin* of the family $F$. From now on, the polytope $P_{origin}(F) \cap P_o(F)$ [2] will be denoted by $P_{dom}(F)$. Within the context of the non-overlapping constraint, we associate to each object to place a given family of polytopes $F$, where each polytope corresponds to one possible positioning of the object. As the ranges of the variables of $\langle O_1, O_2, \ldots, O_d \rangle$ get more and more restricted, the number of distinct elements of $F$ will decrease until it becomes a single fixed polytope, which is associated to the final positioning of the shape $P_{shape}(F)$.

**Definition 10** *extremum polytopes of a family of polytopes*

The *extremum polytopes* of a family $F$ of polytopes is a set of fixed polytopes generated by fixing the origin of $P_{shape}(F)$ to one of the vertices $X_1, \ldots, X_k$ of $P_{dom}(F)$. The *i*-th extremum polytope of $F$ is $X_i + P_{shape}(F)$, it is denoted $\text{Extremum}_i(F)$.

Fig. 4 provides an example of a family $F$ of polytopes described by the shape polytope $P_{shape}(F)$ of vertices $\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}3\\2\end{pmatrix}, \begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}-3\\2\end{pmatrix}, \begin{pmatrix}-4\\1\end{pmatrix}, \begin{pmatrix}-4\\-1\end{pmatrix}$, by the fixed polytope $P_{origin}(F)$ of vertices $\begin{pmatrix}-5\\-1\end{pmatrix}, \begin{pmatrix}-1\\-1\end{pmatrix}, \begin{pmatrix}2\\1\end{pmatrix}, \begin{pmatrix}-4\\3\end{pmatrix}$ and by the tuple of domain variables $\langle O_1, O_2 \rangle$ such that $\underline{O_1} = -6$, $\overline{O_1} = 6$, $\underline{O_2} = -3$, $\overline{O_2} = 7$. Part (A) gives 3 feasible instances $I_1, I_2, I_3$ of the family, while part (B) presents the 4 extremum polytopes $E_1, E_2, E_3, E_4$ associated to $F$.

## 3 The Overlapping Polytope

The purpose of this section is to characterize the portion of the placement space, called the *overlapping polytope*, where positioning the origin of a polytope will necessarily cause it to intersect with another not yet completely fixed polytope.

**Theorem 1**

Let $F$ be a family of polytopes of a $IR^d$ defined by $P_{dom}(F)$ and $P_{shape}(F)$, and let $P$ be a fixed polytope of $IR^d$. If $P$ overlaps[3] all the extremum polytopes of the family $F$, then $P$ overlaps all the members of $F$.

---

[2] Since $P_{origin}(F) \cap P_o(F)$ is the intersection of two polytopes it is also a polytope.

[3] Overlap refers to the definition of *intersection of fixed polytopes* introduced by Definition 8.

The proof of Theorem 1 is given in [3]. When the intersection of all extremum polytopes of a family $F$ is not empty, then one can observe that this intersection coincides with the notion of *compulsory part* introduced in [7]. The *compulsory part* is the portion of space that is covered by all the members of the family $F$.

**Definition 11** *shadow polytope*

The *shadow polytope* of a fixed polytope $P_1$ of $IR^d$ according to a shape polytope $P_2$ of $IR^d$ is a fixed polytope $P_{12}$ of $IR^d$ defined as follows. We consider all the fixed instances $I_{12}$ of $P_2$ such that one vertex of $P_2$ coincides with one vertex of $P_1$. The *shadow polytope*[4] is the convex hull of the origin vertices of all the fixed instances of $I_{12}$. It is denoted $\mathrm{Shadow}(P_1, P_2)$.

One can notice that the shadow polytope of a fixed polytope $P_1$ according to a shape polytope $P_2$ is actually the Minkowski sum [5, pp. 272-279] of $P_1$ and $-P_2$[5].

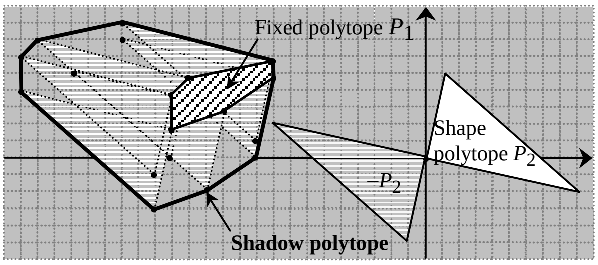



**Fig. 5.** Shadow polytope of $P_1$ according to $P_2$

Fig. 5 shows with a bold line the shadow polytope of the fixed polytope $P_1$ according to the shape polytope $P_2$. It consists of the convex hull of the 18 points that are obtained by making one of the 3 vertices of $P_2$ to coincide with one of the 6 vertices $P_1$.

**Theorem 2**

Let $P_{12}$ be the shadow polytope of a fixed polytope $P_1$ of $IR^d$ according to a shape polytope $P_2$ of $IR^d$.

1° If the origin of $P_2$ is a relative interior point of $P_{12}$, then $P_2$ and $P_1$ overlap.

2° If the origin of $P_2$ is not a relative interior point of $P_{12}$, then $P_2$ and $P_1$ do not overlap.

**Proof of Theorem 2**

Part 1° Suppose $x \in P_{12}^{\bullet}$ then $x \in \left(x^* + (-P_2)\right)^{\bullet}$ for some $x^* \in P_1$.

So $x = x^* + x_2$ where $x_2 \in (-P_2)^{\bullet}$,

hence $x - x_2 \in x + P_2^{\bullet} = (x + P_2)^{\bullet}$ and $x - x_2 = x^* \in P_1$, i.e. $x - x_2 = x^* \in (x + P_2)^{\bullet} \cap P_1$.

---

[4] We call it “shadow” since the shadow polytope is partially looking like the fixed polytope from which it is derived.

[5] We get $-P_2$ by reflecting $P_2$ about its origin.

Now choose $r > 0$ such that $Ball(x^*, r) \subset (x + P_2)^{\bullet}$ and notice that $x^* \in P_1$,

hence $Ball(x^*, r) \cap P_1^{\bullet} \neq \emptyset$, so $(x + P_2)^{\bullet} \cap P_1^{\bullet} \neq \emptyset$.

Part 2° Suppose $P_1^{\bullet} \cap (x + P_2)^{\bullet} \neq \emptyset$ where $x \in bd(P_{12})$,

then there exists an $x_1 \in P_1^{\bullet} \cap (x + P_2)^{\bullet}$.

So $x_1 = x + x_2$ where $x_2 \in P_2^{\bullet}$,

therefore $x = x_1 - x_2 \in P_1^{\bullet} - P_2^{\bullet} \subset (P_{12})^{\bullet}$ which is a contradiction. ❑

**Definition 12** *overlapping polytope*

The *overlapping polytope* of a family of polytopes $F$ of $IR^d$ according to a given shape polytope $P_{shape}$ of $IR^d$ is the polytope (it may be an empty set) defined as follows:

$$\text{Overlapping}(F, P_{shape}) = \bigcap_{i=1}^{|P_{dom}(F)|} \text{Shadow}(\text{Extremum}_i(F), P_{shape}).$$

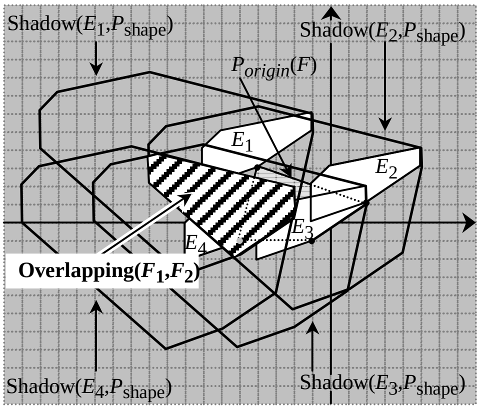



**Fig. 6.** Overlapping polytope

Fig. 6 shows the overlapping polytope of a family of polytopes $F$ according to a shape polytope $P_{shape}$. $P_{origin}(F)$ and $P_{shape}(F)$ respectively correspond to the fixed polytope specified in part (A) of Fig. 2 and to the shape polytope given in part (B) of Fig. 2. $P_{shape}$ is the shape polytope described in the right part of Fig. 5 (i.e. the shape polytope $P_2$). Since $F$ has 4 extremum polytopes $E_1$, $E_2$, $E_3$ and $E_4$, the overlapping polytope is the intersection of the corresponding 4 shadow polytopes. As an easy corollary of Theorems 1 and 2, we have the following theorem.

**Theorem 3**

Let $F$ be a family of polytopes of $IR^d$ and $P_{shape}$ a shape polytope of $IR^d$. For any point $X \in \text{Overlapping}(F, P_{shape})^{\bullet}$ the fixed polytope $X + P_{shape}$ will overlap any fixed polytope of the family $F$.

**Proof of Theorem 3**
From the definition of an overlapping polytope and from Theorem 2, we have that all fixed polytopes $X + P_{shape}$ $\left(X \in \text{Overlapping}\left(F, P_{shape}\right)^{\bullet}\right)$ overlap all extremum polytopes of $F$. From Theorem 1, we generalize to the fact that they overlap all fixed polytopes of $F$. ❑

The overlapping polytope is related to the notion of forbidden region which was introduced in [2]. It is a forbidden portion of the space according to the binary non-overlapping constraint between two families of polytopes. However unlike the forbidden region, it is multi-dimensional and it has a more general shape than a rectangle. In Sect. 4 and 5 we will prune the origin of a polytope in order to avoid that it is a relative interior point of a given overlapping polytope.

## 4 A Filtering Algorithm for the non-overlapping Constraint between Two Convex Polygons

This section first presents a linear algorithm for computing the overlapping polytope. It then gives a filtering algorithm which exploits the previous overlapping polytope in order to prune the origin variables of a polygon.

### 4.1 Computing the Overlapping Polytope in Two Dimensions

Suppose we want to compute the overlapping polytope for a shape polytope $P_{shape}$ according to a family $F$ of polygons.

**Computing the shadow polytope.** Let $Q$ denote the domain polytope $P_{dom}(F)$ and let $w_1,\ldots,w_m$ be the vertices of $Q$ in counter-clockwise order. Since the shadow polytope $P = \text{Shadow}\left(P_{shape}(F), P_{shape}\right)$ is the Minkowski sum of $P_{shape}(F)$ and $-P_{shape}$ it can be computed in linear time in the number of vertices of $P_{shape}(F)$ and $P_{shape}$ by using the algorithm given in [5, page 277] for computing the Minkowski sum in two dimensions.

**Extracting the relevant halfspaces.** If we denote the overlapping polytope by $O$, we have $O = \cap_{j=1}^{m} w_j + P$. If $P$ has $n$ edges, then $P$ is the intersection of $n$ halfspaces $H_1,\ldots,H_n$, where the boundary of $H_i$ contains the $i$th edge (see Fig. 7). And hence, $O = \cap_{i=1}^{n} \cap_{j=1}^{m} w_j + H_i$. If we look at Fig. 7, we observe that the halfspace $w_2 + H_2$ is contained in the halfspaces $w_1 + H_2$ and $w_3 + H_2$. Thus of the three halfspaces only $w_2 + H_2$ has to be considered in the computation of $O$. This observation holds in general: for every $i$ $(1 \le i \le n)$ there is a $j(i)$ such that $w_{j(i)} + H_i \subseteq w_j + H_i$ for $j = 1,\ldots,m$.

We call $w_{j(i)}$ an *extremal vertex* of $Q$ with respect to $H_i$. We have just seen that $O = \cap_{i=1}^{n} w_{j(i)} + H_i$.

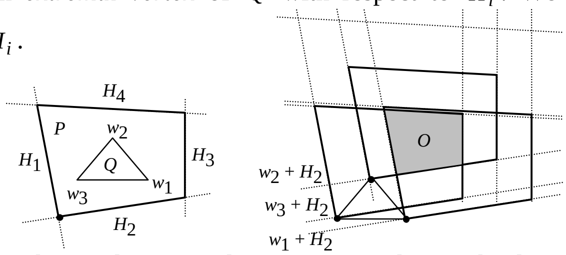



**Fig. 7.** Computing the overlapping polytope $O$ according to the domain polytope $Q$ and the shadow polytope $P$ (the origin of $P$ is the intersection of $H_1$ and $H_2$)

How do we find these extremal vertices efficiently? In two dimensions this is quite easy. Let us look at Fig. 8 and suppose we want to find the extremal vertex for $H_2$. Let $n(H_2)$ denote the normal vector of the edge induced by $H_2$. In two dimensions we define the normal vector of the edge induced by two vertices $u$ and $v$ (given in counter-clockwise ordering) as $n(u,v) = \begin{pmatrix} v_y - u_y \\ u_x - v_x \end{pmatrix}$, i.e. we suppose that normal vectors point to the outside. In order to find an extremal vertex for $H_2$, we perform a parallel slide of $H_2$ in direction $-n(H_2)$; the boundary of $H_2$ hits the vertices of $Q$ in the order $w_1, w_3, w_2$. And since $w_2$ is the last vertex to be hit, it is the extremal vertex. When is $w_2$ extremal with respect to some halfspace $H_i$? Let $e$, $f$ denote the edges incident to $w_2$. Obviously, $w_2$ is extremal when $-n(H_i)$ lies in the cone spanned by the normal vectors $n(e), n(f)$, as shown on the right hand side of Fig. 8.

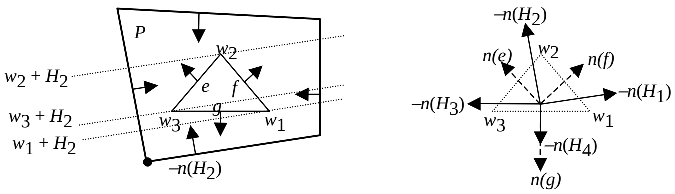



**Fig. 8.** Finding the extremal vertices of a polygon $Q$ for the halfspaces induced by the edges of $P$ (the dashed lines indicate the translations of $H_2$ which intersect a vertex of $Q$). The right hand side shows the respective cones of the vertices $w_1$, $w_2$, $w_3$.

In order to decide whether a vector $\vec{d}$ lies in a given cone, we define $angle(\vec{d})$ as the counter-clockwise angle between the positive *x*-axis and $\vec{d}$. Then we can easily perform the *in-cone-test* by comparing the angles of $\vec{d}$ and the vectors that are spanning the cone. Suppose that $w_1$ is the vertex of $Q$ with largest *x*-coordinate and, in case of tie, smallest *y*-coordinate. If we start in $w_1$ and visit the edges of $Q$ in counter-clockwise ordering the angles of the normal vectors increase monotonously in the interval $[0;2\pi[$. A similar observation can be made for the negative normal vectors for the edges (or halfspaces) of $P$. And hence determining the extremal points for the halfspaces of $P$ amounts to a merging of angles. This leads to Algorithm 1 for which the runtime is clearly in $O(n+m)$.

We want to point out that it is not necessary to compute the angles explicitly in order to do the comparison. Suppose we want to compare the angle between two directions $\vec{d} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$ and $\vec{e} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$. First we compare the quadrants, where $quadrant(\vec{d})$ is 1 if $d_1 \geq 0 \wedge d_2 \geq 0$, 2 if $d_1 < 0 \wedge d_2 \geq 0$, 3 if $d_1 < 0 \wedge d_2 < 0$, 4 if $d_1 \geq 0 \wedge d_2 < 0$. If the quadrant of $\vec{d}$ is greater (smaller) than the quadrant of $\vec{e}$ then the angle is also greater (smaller). If the quadrants are equal we know that there is an acute angle between the two directions. And hence we can consider the third component of the cross product $\vec{d} \times \vec{e}$, which is $d_1 e_2 - d_2 e_1$. When the quadrants are equal, this component is negative iff $angle(\vec{d}) > angle(\vec{e})$.

**Input :** Polygons $P$=( $v_1,\ldots,v_n$) and $Q$=( $w_1,\ldots,w_m$).
**Require:** The vertices are in counter-clockwise ordering. The vertex $v_1$ has smallest $x$-coordinate and, in case of ties, largest $y$-coordinate; vertex $w_1$ has largest $x$-coordinate and, in case of ties, smallest $y$-coordinate.

$v_{n+1} \leftarrow v_1$; $w_{m+1} \leftarrow w_1$; $i \leftarrow 1$; $j \leftarrow 1$;
**repeat**
    **while** $angle(-n(v_i, v_{i+1})) > angle(n(w_j, w_{j+1}))$ **do** $j \leftarrow j+1$;
    store $w_j$ as an extremal vertex of $H_i$;
    $i \leftarrow i+1$;
**until** $i=n+1$;

**Alg. 1.** Computing extremal vertices

**Computing the intersection of the relevant halfspaces.** Now we can compute $O = \cap_{i=1}^{n} w_{j(i)} + H_i$. It is well known that this can be done in time $O(n \log n)$ [5, page 71]. But we can provide an $O(n)$ algorithm since we recognize that $angle(H_i) < angle(H_{i+1})$ for $i = 1,\ldots,n-1$. Our algorithm computes the intersection of the halfspaces iteratively; in the $i$-th iteration $(i \geq 2)$ we compute $O_i = \cap_{k=1}^{i} w_{j(k)} + H_k$. We represent $O_i$ with a data structure $\mathcal{B}_i$ describing its boundary. The boundary of the halfspace $w_{j(k)} + H_k$ is the line $L_k = \{(w_{j(k)} + v_k) + \lambda(v_{k+1} - v_k); \lambda \in IR\}$. The boundary of $O_i$ may be infinite, and then it consists of two rays and of zero or more line segments. If it is finite, it consists only of line segments. We call such a ray or a line segment a *boundary element* and $\mathcal{B}_i$ will be a list of boundary elements.

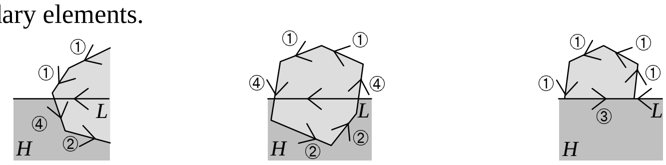



**Fig. 9.** Intersection of the halfspace $H$ (with the bounding line $L$) with an infinite or a finite boundary

Now suppose that $O_{i-1}$ is not empty and we have computed the list $\mathcal{B}_{i-1}$. In order to compute $\mathcal{B}_i$ we have to determine how the boundary changes if we add the halfspace $H = w_{j(i)} + H_i$ to the intersection. It is clear that the halfspace can contribute at most one new boundary element, but some of the old elements may have to be updated or discarded. Let us consider an old element $B$ from $\mathcal{B}_{i-1}$ and distinguish four possible cases (In Fig. 9 the respective case is marked beside every element):

1. $B$ lies to the right of $L_i$: then $B \cap L_i = \emptyset$ and we can discard $B$.
2. $B$ lies to the left of $L_i$: then $B \subset H$ and we keep $B$ unchanged.
3. $B$ lies on $L_i$: this means that the normal vector of $H$ and of the halfspace form which $B$ originates are anti-parallel. And hence the interior of $O_i$ is empty and $\mathcal{B}_i$ only consists of $B$.
4. $B$ and $L_i$ properly intersect in a point $x$: then we have to update $B$; we discard the part of $B$ which lies to right of $L_i$.

It is easy to find the contribution $B_H$ of $H$ to the boundary. Since the boundary is convex there can be at most two proper intersection points. If there are two intersection points $x$ and $x'$ then $B_H$ is the line segment between $x$ and $x'$. In case there is only one point $x$, $B_H$ is a ray starting in $x$. If all elements of $\mathcal{B}_{i-1}$ lie to the right of $L_i$, then the intersection is empty and we can terminate. If all old boundary elements lie to the left of $L_i$ then $H$ is redundant, i.e. it does not contribute to the boundary.

In the *i*-th iteration we first update $\mathcal{B}_{i-1}$ as just discussed and append the contribution of $w_{j(i)} + H_i$ to the end of the list, if there is any. Thus we obtain the new list $\mathcal{B}_i$. In order to obtain the desired time bound we cannot afford to test $L_i$ against all old boundary elements. Suppose $\mathcal{B}_{i-1} = \langle B_1, \ldots, B_l \rangle$ and that $B_\lambda$ originates from $H_{h(\lambda)}$. From the construction of $\mathcal{B}_{i-1}$ it is easy to see that $h(1) < \ldots < h(l)$. And hence the angles of the negative normal vectors of $B_1, \ldots, B_l$ increase monotonously and are smaller than the angle of $-n(H_i)$. Thus we can do the test in the following way. First we process the list from left to right and discard elements lying to the right of $L_i$ until we find an element that does not lie to the left of $L_i$, then we process the list from right to left and do the same. If the list becomes empty, we know that the intersection is also empty. Due to the order of the elements in $\mathcal{B}_{i-1}$ we can be sure that all elements that we do not test lie to the left of $L_i$ and hence need no update.

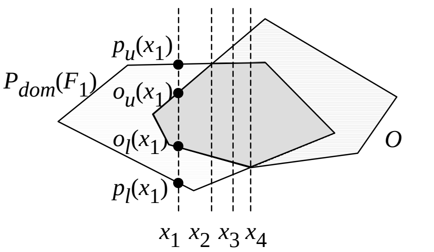
Our algorithm performs only $O(n)$ tests altogether. This can be seen as follows. Assume we test a boundary element $B$ and a line $L_i$. If $B$ lies to the right of $L_i$ we charge the test to $B$, otherwise we charge it to $L_i$. Every line $L_i$ is charged at most twice, and every boundary element is charged at most once, because it is immediately discarded after being charged. This gives us the desired bound.

### 4.2 Pruning in Two Dimensions

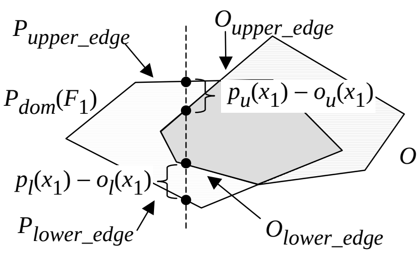
Suppose we want to prune the origin of a family $F_1$ with respect to a family $F_2$. We describe the algorithm for the domain variable $O_x$ which denotes the *x*-coordinate of the origin of $F_1$. In the previous section we have seen how to compute

$O = \text{Overlapping}(F_2, P_{shape}(F_1))$. We know that we have to place the origin of $F_1$ into $P_{dom}(F_1) \setminus O^{\bullet}$. Let $L_{x_0}$ denote the vertical line given by the equation $x = x_0$. We can prune a value $x_0$ from $O_x$ if the set $I(x_0) = (P_{dom}(F_1) \setminus O^{\bullet}) \cap L_{x_0}$ contains no point with integer coordinates. The line $L_{x_0}$ can intersect the boundary of the polygon $P_{dom}(F_1)$ in at most two points. Let $p_l(x_0)/p_u(x_0)$ denote the *y*-coordinate of the lower/upper intersection point (see part (A) of Fig. 10), if there is no intersection set $p_l(x_0)$ to $\infty$ and $p_u(x_0)$ to $-\infty$. And define $o_l(x_0)/o_u(x_0)$ in an analogous way for $O$. Suppose $\min_x(P_{dom}(F_1)) \le x \le \max_x(P_{dom}(F_1))$. Then $I(x_0)$ is empty iff $o_l(x_0) < p_l(x_0)$ and $p_u(x_0) < o_u(x_0)$. And for integral $x_0$ the set $I(x_0)$ contains a point with integer coordinates iff there is an integer $k$ with $p_l(x_0) \le k \le p_u(x_0)$ and $k \le o_l(x_0) \vee k \ge o_u(x_0)$. This observation leads to the following algorithm. We (conceptually) move a sweep-line [8, pp. 10-11] $L$ from left to right: we start in $\underline{O}_x$ and stop in $\overline{O}_x$.

**A. Pruning in the continuous case.** We first describe an algorithm which does not achieve maximum pruning, because it does not remove $x_0$ from $O_x$ if $I(x_0)$ contains no integer point, but only if $I(x_0)$ is empty. In order to do so it suffices to find the *x*-coordinates where one of the differences $o_l(x_0) - p_l(x_0)$ and $p_u(x_0) - o_u(x_0)$ changes its sign. This can only happen if there is a *proper intersection*[6] between two lower edges or two upper edges of the two polygons (see transitions $x_1 \to x_2 \to x_3$ in part (A) of Fig. 10) or a vertex of one polygon lies on the boundary of the other one (see sweep-line at $x_4$ in part (A) of Fig. 10).



(A) Different positions of the sweep-line



(B) Sweep-line status

**Fig. 10.** Illustration of the sweep

**Sweep-line status.** We restrict our attention to the case where $\min_x(P_{dom}(F_1)) < x_0 < \max_x(P_{dom}(F_1))$ and $\min_x(O) < x_0 < \max_x(O)$. Then the sweep-line intersects both polygons in two points and it does not intersect a vertical edge. The data structure representing the sweep-line status [5 page 68] stores its current position $x_0$, the signs of the differences $o_l(x_0) - p_l(x_0)$ and $p_u(x_0) - o_u(x_0)$ and the four edges which are

[6] We call an intersection between two edges *proper* if they intersect in a single point which is not an endpoint of either edge.

intersected by it: $P_{lower\_edge}$, $P_{upper\_edge}$, $O_{lower\_edge}$, $O_{upper\_edge}$ (see part B of Fig. 10). If the sweep-line intersects a vertex $v$ of a polygon, we store the edge starting at $v$, i.e. the edge where the opposite vertex lies to the right of $L_{x_0}$.

**Events.** An event is an *x*-coordinate where the sweep-line status has to be updated. As we said before, this is the case whenever the sweep-line hits a vertex or a proper intersection point between lower or upper edges. Since the sweep-line intersects only 4 edges, we can always determine the next event in constant time without maintaining any additional data structure. Processing an event can also be done in constant time. Note that there may be several updates to the sweep-line status at a single event. For every edge of either polygon there can be at most two proper intersection points. Hence every edge gives rise to a constant number of events. If $n$ denotes the number of edges of $O$ and $m$ the number of edges of $P_{dom}(F_1)$, the overall sweep can be done in time $O(n+m)$. The additional time needed for pruning depends on the representation of a domain variable.

**B. Additional pruning in the discrete case.** Now suppose that we want to achieve some stronger pruning, taking into account the fact that $O_y$ will be an integer. We can prune a value $x_0 \in O_x$ not only if $I(x_0)$ is empty but also if $I(x_0)$ does not contain a vertex with integer coordinates. One way to do this is to generate *check events* which make the sweep-line stop at every $x_0$ in $O_x$ (in addition to the *regular events* generated in the continuous case) and to check in constant time whether $x_0$ can be pruned. This increases the running time by $O(|O_x|)$.

One does not have to generate all check events. If $I(x_0)$ is empty at some regular event, then there is no need to generate check events until the next regular event occurs. And if at least one of the differences $p_u(x_0) - o_u(x_0)$ or $o_l(x_0) - p_l(x_0)$ is greater than or equal to 1 at some event $x_0$ and will not go below 1 until the next regular event $x_1$, then we know that $I(x)$ contains an integer point for every integer $x$ in $[x_0, x_1]$, and hence we do not have to generate check events. So check events are only necessary if both upper and both lower edges are close together.

### 4.3 Summary of the Filtering Algorithm

We are given two families $F_1$ and $F_2$ of polygons. Let $n_i$ and $m_i$ denote the number of vertices of the shape and origin polygon of family $F_i$ respectively. We do the following to prune the origin variables $O_{1,x}$ and $O_{1,y}$ of $F_1$ according to $F_2$:

- Compute $P_{dom}(F_1) = P_{origin}(F_1) \cap P_o(F_1)$. This can be done in time $O(m_1)$ using the algorithm given in [9] and yields a polygon with at most $m_1 + 4$ vertices,
- Compute the overlapping polytope $O = \text{Overlapping}(F_2, P_{shape}(F_1))$. This involves the following three substeps:
  – compute $P$ as the Minkowski sum of $P_{shape}(F_2)$ and $-P_{shape}(F_1)$ in time $O(n_1 + n_2)$,

– find for every facet of $P$ an extremal vertex of $P_{dom}(F_2)$, which requires time $O(n_1+n_2+m_2)$,

– compute $O$ as the intersection of $n_1+n_2$ halfspaces in linear time.

- Prune $O_{1,x}$ and $O_{1,y}$ with the sweep algorithm described previously in time $O(n_1+n_2+m_1)$ or $O\left(n_1+n_2+m_1+\left|O_{1,x}\right|+\left|O_{1,y}\right|\right)$ if we want to take into account the fact that the coordinates are integer.

## 5 A Filtering Algorithm for the non-overlapping Constraint between Two *d*-dimensional Boxes

This section develops an efficient filtering algorithm for the special case where we have *d-dimensional boxes*. A *d*-dimensional box of origin $\langle O_1,O_2,\ldots,O_d\rangle$ and size $\langle S_1,S_2,\ldots,S_d\rangle$ where $O_1,O_2,\ldots,O_d$ are domain variables and $S_1,S_2,\ldots,S_d$ are strictly positive integers is a family of polytopes such that:

– the shape of the family is defined as the convex hull of the following $2^d$ vertices of coordinates $s_1,s_2,\ldots,s_d$ where $s_i$ $(i=1,2,..,d)$ stands for $0$ or for $S_i$,

– the initial possible placement for the origin of the previous shape is defined by $box(O_1,O_2,\ldots,O_d)$,

– $\langle O_1,O_2,\ldots,O_d\rangle$ is the origin of the family of polytopes.

Consider now two *d*-dimensional boxes $B_1$, $B_2$ of respective origins $\langle O_{11},O_{21},\ldots,O_{d1}\rangle$, $\langle O_{12},O_{22},\ldots,O_{d2}\rangle$ and respective sizes $\langle S_{11},S_{21},\ldots,S_{d1}\rangle$, $\langle S_{12},S_{22},\ldots,S_{d2}\rangle$. We describe how to prune the origin of $B_2$ according to $B_1$. The overlapping polytope of $B_1$ according to $\langle S_{12},S_{22},\ldots,S_{d2}\rangle$ is defined by all the points of coordinates $p_1,p_2,\ldots,p_d$ such that, for all $i\in 1..d$ we have $\overline{O_{i1}}-S_{i2}\le p_i\le \underline{O_{i1}}+S_{i1}$. Pruning the origin of $B_2$ according to $B_1$ consists preventing the origin of $B_2$ from being a relative interior point of the previous overlapping polytope. For this purpose we count the number of times the condition $\overline{O_{i1}}-S_{i2}+1\le \underline{O_{i2}}\wedge\overline{O_{i2}}\le\underline{O_{i1}}+S_{i1}-1\quad(1\le i\le d)$ holds. The non-overlapping constraint fails if the previous condition holds $d$ times. If it holds for all dimensions except one dimension $j$, then we remove the interval of values that starts at $\overline{O_{j1}}-S_{j2}+1$ and ends at $\underline{O_{j1}}+S_{j1}-1$ from the domain variable $O_{j2}$. This leads to an algorithm for which the runtime is clearly in $O(d)$.

## 6 Discussion and Conclusion

Note that it should be easy to apply the results of this paper for handling other binary geometrical constraints for which the forbidden set of points is convex. As an example, consider the case of the non-inclusion constraint which enforces a first given polytope to not be completely included within a second given polytope. The only change is to provide an algorithm that computes the shadow polytope according to this new type of constraint. Fig. 11 shows the shadow polytope $S$ (see part B) associated to the non-inclusion constraint between polytope $P_1$ (see part A) and polytope $P_2$ (see part B). This indicates that vertex $O$ of polytope $P_1$ should not be located within polytope $S$ (see parts C and D), if $P_1$ should not be completely included within $P_2$.

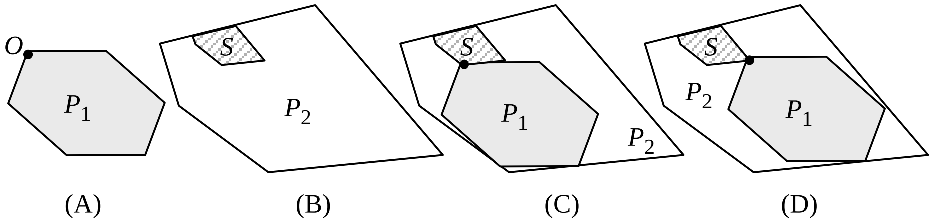



**Fig. 11.** Example of another convex constraint: the non-inclusion constraint

We have introduced necessary conditions for the non-overlapping constraint between polytopes. The key idea that leads to the propagation algorithm is that one can derive the overlapping polytope by considering only a very restricted number of instances of a family, namely the extremum polytopes. From these necessary conditions, we have derived efficient filtering algorithms for the non-overlapping constraint between two convex polygons as well as the non-overlapping constraint between two *d*-dimensional boxes [1]. However if we would like to come up with a more efficient propagation algorithm for the case of a clique of non-overlapping constraints, the following question remains open. One would get much more propagation by aggregating the different overlapping polytopes, but it is not clear how to efficiently generalize the algorithm presented in [2] to this situation.

## References


1. Beldiceanu, N., Contejean, E.: Introducing global constraint in CHIP. *Mathl. Comput. Modelling* Vol. 20, No. 12, 97-123, 1994.
2. Beldiceanu, N.: Sweep as a generic pruning technique. In *TRICS: Technique foR Implementing Constraint programming*, CP2000, Singapore, 2000.
3. Beldiceanu, N., Guo, Q., Thiel, S.: Non-overlapping Constraint between Convex Polytopes. SICS technical report T2001-12, (May 2001).
4. Berger, M.: *Geometry II*, Chapter 12. Springer-Verlag, 1980.
5. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: *Computational Geometry – Algorithms and Applications*. Springer, 1997.
6. Chamard, A., Deces, F., Fischler, A.: A Workshop Scheduler System written in CHIP. *2nd Conf Practical Applications of Prolog*, London, April 1994.
7. Lahrichi, A., Gondran, M.: Théorie des parties obligatoires et découpes à deux dimensions. Research report HI/4762-02 from EDF (Électricité de France), (23 pages), 1984. In French.
8. Preparata F.P., Shamos M.I.: *Computational Geometry. An Introduction.* Springer-Verlag, 1985.
9. Sutherland, I.E., Hodgman, G.W.: Reentrant Polygon Clipping, *CACM*, 17(1), 32-42, 1974.
10. Van Hentenryck, P.: *Constraint Satisfaction in Logic Programming.* The MIT Press, 1989.