On the Hyperbox – Hyperplane Intersection Problem

CARLOS LARA
JUAN J. FLORES
FELIX CALDERON

Division de Estudios de Posgrado
Facultad de Ingenieria Electrica
Universidad Michoacana
Edificio DEP-FIE, Ciudad Universitaria
58060, Morelia, Mexico
(larac,juanf,calderon)@umich.mx

Abstract. Finding the intersection between a hyperbox and a hyperplane can be computationally expensive specially for high dimensional problems. Naive algorithms have an exponential complexity. A border node is a node (in the graph induced by the hyperbox) at or next to the intersection of the hyperbox and the hyperplane. The algorithm proposed in this paper implements a systematic way to efficiently generate border nodes; given a border node, a subset of its incident edges is explored to determine one or more intersections. This systematic exploration allows us to focus on the border region, discarding the two regions before and after the plane. Pruning those regions produces a computational cost linear on the number of vertices of the hyperpolygon that represents the intersection.

Keywords: hyperbox, hyperrectangle, hyperplane, vector, graph searching

(Received May 15, 2009 / Accepted August 11, 2009)

1 Introduction

The problem we are addressing in this paper is to determine the intersection between a hyperbox[1] and a hyperplane. A simple example of the intersection between a cube and a plane is shown in Figure 1. The intersection of box B and plane P describes a polygonal region that is a subset of P. This region can be represented by the sequence of points $c_1, \ldots, c_6$ (i.e. the vertices of the intersection polygon). As can be seen in Figure 1, the vertices of the intersection polygon lie on the edges of Box B (perhaps on vertices at the end of edges). A naive algorithm [1, 7] to find the points c_i takes each edge and finds the point that intersects the plane with that edge (if any).

The general problem consists of finding the intersection between a hyperbox and a hyperplane. A hyperbox is a generalization of the rectangle to n dimensions. A hyperplane in the same space has $n-1$ dimensions. Since the number of edges of a hyperbox is $n2^{n-1}$ [5, 2], the computational cost of the naive algo-

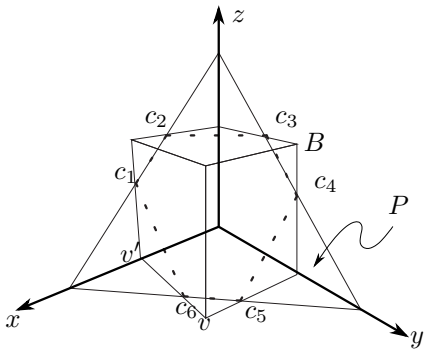


Figure 1: Intersection between a box B and a plane P

[1] A hyperbox is an n-dimensional body similar to a hypercube, except that its sides are not necessarily of the same length

rithm is exponential with respect to the problem's dimensionality.

Hyperboxes appear in the mathematical programming formulation of many combinatorial optimization problems, in which solutions can be represented by binary vectors [10]. In such formulation, the decision variables can assume values in the interval $[0, 1]$ and correspond to the solution components; hyperplanes correspond to constraints, and the hyperbox – hyperplane intersection represents feasible solutions [1]. The intersection between a hyperbox B and a hyperplane P describes a hyperpolygon. The problem addressed by this paper is to compute the vertices of that hyperpolygon.

This paper is organized as follows, Section 2 presents related work. Section 3 introduces some basic definitions; Section 4 states the hyperbox–hyperplane intersection problem; Section 5 introduces a naive solution to solve the problem; Section 6 introduces the proposed algorithm; Section 7 discusses the experimental results; finally, Section 8 concludes our work.

2 Related Work

Detecting whether two geometric objects intersect and computing the region of intersection are fundamental problems in computational geometry. Geometric intersection problems can be found in a number of applications: geometric packing and covering, wire and component layout in VLSI, map overlay in geographic information systems, motion planning, collision detection, etc.

There are two main streams of work related to the formulation of the algorithm presented in this paper. The first one deals with the combinatorial problem of counting how many vertices or corners there are in the convex hyperpolygon resulting of the intersection of a hyperbox and a hyperplane. The second one deals with the problem of determining the exact location of all corners of the intersection.

In the first stream, Harary et. al. [4], present a comprehensive survey of the theory of hypercube graphs. Although their paper does not contribute to the solution of our problem, their compilation of concepts is insighting. Sykora and Vrto [8] provide tight bounds for the number of crossings of a hypercube and cube connected cycles. This problem has important applications in printed circuit board layout, VLSI circuit routing, and automated graph drawing (see also [9]). Mount[6] solves a number of intersection problems which mainly involves 2D polygons.

In the second stream, and more related to our work, Rezk and Kolb [7] provide a solution algorithm for computing the intersection of a box with a plane in 3D. Their motivation leads to the field of computer graphics and 3D rendering. Given that their application deals only with 3D objects, their work was not extended to intersections in n dimensions. Instead of providing a general algorithm, authors analyze all possible scenarios for the 3D case. With respect to the general problem of determining the position of all corners of the intersection of a hypercube with a hyperplane, at the moment of writing this paper, to the best knowledge of its authors, there are no publications providing a general algorithm to solve it.

3 Basic Definitions

A *closed hyperbox* (or hyperrectangle) in an n–dimensional real space is defined as the cartesian product of n intervals, that is

$$B = [a_1, b_1] \times \ldots \times [a_n, b_n] \subset \mathbb{R}^n \tag{1}$$

A point $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is inside hyperbox B if $\forall i \in \{1, \ldots, n\}\, a_i \leq x_i \leq b_i$. The values a_i and b_i are the *lower* and *upper* limits of the i–th dimension of B, respectively. $V(B)$ is the set of vertices of Hyperbox B; a point $x = (x_1, \ldots, x_n)$ is a *vertex* of B (i.e. $x \in V(B)$), if $\forall i \in \{1, \ldots, n\}\, x_i \in \{a_i, b_i\}$ [4].

Function L associates a label l to every vertex $v \in V(B)$, such that $l_i = 0$ if $v_i = a_i$ and $l_i = 1$ if $v_i = b_i$; from this point on, we will refer to a vertex by its vertex representation v or by its associated label $L(v)$. u and v are adjacent vertices of the graph induced by the hyperbox B if they differ exactly in one position. That is, if $p = L(u)$ and $q = L(v)$, and $\exists! j \in \{1, \ldots, n\}\, p_j = \overline{q}_j$ and $\forall i \neq j\ p_i = q_i$; $[u, v]$ is an edge of that graph.

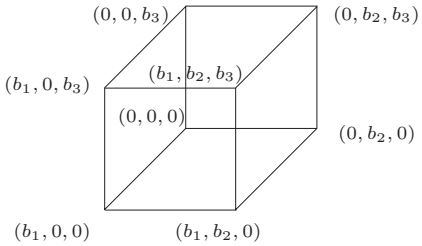
A *k–dimensional hyperplane* can be defined in an n–dimensional space by a set of $(n-k)$ non-degenerate linear equations. Consequently, an $(n-1)$–*hyperplane* or simply *hyperplane* is defined by a single linear equation. In the following we define an hyperplane by the linear equation[2]

$$\sum_{i=1}^{n} x_i = \alpha. \tag{2}$$

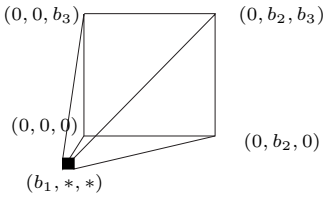
Definition 1. *A hyperbox B is **located at the origin** if $\forall i \in \{1, \ldots, n\}\, a_i = 0$. That is, the hyperbox has the form $B = [0, b_1] \times \ldots \times [0, b_n]$; where $(b_1, \ldots, b_n)$ is the upper corner of B.*

Definition 2. *An **ordered hyperbox** is a hyperbox at the origin such that the coordinates of its upper corner satisfy $b_1 \leq b_2 \leq \ldots \leq b_n$.*

[2] A generalization is discussed in section 6.3



(a) A hyperbox



(b) Grouping nodes that start with b_1

Figure 2: A single s–face represents several vertices

Our approach consists of processing vertices of a hypercube instead of its edges. There are 2^n vertices, then we are interested on properties of vertex subsets that avoid time complexity. The vertex set representation used in this paper is related to the concept of squashed cubes introduced by Graham and Pollak in [3]. Consider the example of Figure 2. The four vertices that start with b_1, i.e. $\{(b_1,0,0),(b_1,0,b_3),(b_1,b_2,0),(b_1,b_2,b_3)\}$ (Figure 2(a)) are represented by the s–face $(b_1,*,*)$ (Figure 2(b)).

Definition 3. *A **squashed face** or **s–face** of an n–dimensional hyperbox B is an n–tuple $s=(s_1,\ldots,s_n)$ where $s_i \in \{0,1,*\}$. We say that s_i is defined if $s_i \in \{0,1\}$, it is undefined if $s_i = *$. An s–face s represents a set $V(s) \subseteq V(B)$ whose elements are those vertices obtained by replacing every undefined variable by 0 or 1. If $s=(b_1,\ldots,b_j,*,\ldots,*)$, the cardinality of $V(s)$ is $|V(s)| = 2^{n-j}$. Just like every vertex has an associated label, the label associated to an s–face is s itself.*

4 Problem Statement

Let $B = [a_1,b_1] \times \ldots \times [a_n,b_n] \subset \mathbb{R}^n$ be an n–dimensional hyperbox, and P the hyperplane satisfying the constraint $\sum_{i=1}^{n} x_i = \alpha$. Clearly, P has $n-1$ dimensions. Hyperbox B and hyperplane P intersect at a region that is a subset of P. That region is a hyperpolygon on $n-1$ dimensions.

The problem addressed by this paper is to compute the intersection between a hyperbox B and a hyperplane P. There may be more than one way to represent a hyperpolygon; the one we have chosen is by its vertices (i.e. $C = \{c_i \mid i = 1,\ldots,m\}$, see Figure 1). The vertices of the intersection polygon lie on the edges of the hyperbox (perhaps on vertices at the end of edges).

Since the number of edges on an n–dimensional hyperbox is $n2^{n-1}$, checking every edge is not an efficient option. This paper presents an algorithm to determine all vertices of the intersection on time $O(nm)$, where m is the number of vertices of the intersection polygon.

5 Naive Algorithm

This approach presented in [1, 7], uses a parametric representation for an edge. That is, given two adjacent vertices $v=(v_1,\ldots,v_n)$ and $u=(u_1,\ldots,u_n)$ of the hyperbox, the edge e_{uv} is given by

$$e_{uv}(t) = v + t(u-v) \tag{3}$$

where $t \in [0,1]$. When the edge intersects the hyperplane then the components of e_{uv} sum up to α. Replacing t by $t^\star$ in Equation 3 and solving for $t^\star$

$$t^\star = \frac{\alpha - \sum_{i=1}^{n} v_i}{\sum_{i=1}^{n}(u_i - v_i)} \tag{4}$$

If $t^\star \in [0,1]$ there exists an intersection between edge e_{uv} and the hyperplane α.

Algorithm 1 traverses all edges; if an intersection is found, the corresponding vector $e_{uv}(t^\star)$ is added to list $\mathcal{L}$, containing all results.

Algorithm 1 Naive(B, α)

Input: A hyperbox B and the hyperplane defined by α
Output: $\mathcal{L}$, the list of solutions for the intersection of B and α

```
1: L ← {}
2: Compute the edges E(B)
3: for all e_uv ∈ E(B) do
4:     Compute t* with Equation 4
5:     if 0 ≤ t* ≤ 1 then
6:         L ← L ∪ e_uv(t*)
7:     end if
8: end for
9: return L
```

6 Proposed Algorithm

This section describes the proposed algorithm to compute the intersection of hyperplane P and hyperbox B. For simplicity, to explain the algorithm we consider the

case of an ordered hyperbox located at the origin. Generalizations of this case are discussed in section 6.3. There are some preliminary definitions that need to be stated, and which will lead in our way to describing the intersection algorithm.

Any intersection point $c \in C$ can be found by locating an adjacent vertex to c. As an example, the intersection c_6 in Figure 1 can be reached from one of the two adjacent vertices v or v'. Vertex v is over the plane α (is a high border vertex) while v' is under the plane (is a low vertex) . A **border vertex** is a vertex under the plane from which one or more solutions can be reached. Since any intersection point is restricted by Equation 2, a border vertex is defined formally as:

Definition 4. *A node $v = (v_1, \ldots, v_n)$ is a **low border vertex** or simply **border vertex** if $\exists k, v_k = 0 \wedge \sum_{i=1}^{n} v_i \leq \alpha < b_k + \sum_{i=1}^{n} v_i$.*

Definition 5. *A node $w = (w_1, \ldots, w_n)$ is a **high border vertex** if $\exists k, w_k = b_k \wedge \sum_{i=1}^{n} w_i \geq \alpha > (\sum_{i=1}^{n} w_i) - b_k$.*

We select low border vertices (border vertices) to find intersections, another option is to use high border vertices. The two approaches are equivalent: when low border vertices are used, a quantity is added to one dimension to reach the intersection point; and when high border vertices are used, a quantity is subtracted to reach the intersection point. Next section discusses how to take advantage of both options.

Lemma 6.1. *Let B be an ordered hyperbox, and $s = (v_1, \ldots, v_j, *, \ldots, *)$ a squashed face with $j < n$ defined variables. If*

$$\sum_{i=1}^{j} v_i \leq \alpha < b_{j+1} + \sum_{i=1}^{j} v_i \tag{5}$$

then $\exists! v \in V(s)$ $\sum_{i=1}^{n} v_i \leq \alpha$. In other words, s has a unique border vertex.

Proof. Let be $v = (v_1, \ldots, v_j, 0, \ldots 0) \in V(s)$, then $\sum_{i=1}^{n} v_i = \sum_{i=1}^{j} v_i \leq \alpha$. Also let $Y = V(s) \setminus \{v\}$, then $\forall y \in Y$ $\sum_{i=1}^{j} y_i > \alpha$ because $\forall y = (y_1, \ldots, y_n) \in Y, \exists k \in \{(j+1), \ldots, n\}$, $y_k = b_k$, $b_{j+1} \leq \ldots \leq b_n$, for an ordered hyperbox and $\alpha < b_{j+1} + \sum_{i=1}^{j} v_i$ so $\alpha < b_k + \sum_{i=1}^{j} v_i$. □

Lemma 6.1 is very useful to find solutions in a systematic manner, but in general an s–face does not satisfy the condition expressed by Equation 5. Exploring the 2^{n-j} nodes in $V(s)$ to find solutions can be computationally expensive.

Our strategy consists of partitioning $V(s)$ in such a way that at least one component of this partition $V((v_1, \ldots, v_k, *, \ldots, *)) \subset V((v_1, \ldots, v_j, *, \ldots, *))$ (where $j < k$) satisfies Equation 5. On the other hand, it is desirable for the partition to have a small number of components. The following lemma states that any s–face $(v_1, \ldots, v_j, *, \ldots, *)$ can be partitioned into $k - j + 1$ components, containing the squashed face $(v_1, \ldots, v_k, *, \ldots, *)$.

Lemma 6.2. *Let $s = (v_1, \ldots, v_j, *, \ldots, *)$ and $s' = (v_1, \ldots, v_k, *, \ldots, *)$ be two s–faces where $j < k$ such that $V(s') \subset V(s)$ then*

$$\mathcal{P}(s, s') = V((v_1, \ldots, v_k, *, \ldots, *)) \cup \bigcup_{i=j+1}^{k} V((v_1, \ldots, v_{i-1}\overline{v}_i, *, \ldots, *))$$

is a partition of $V(s)$.

Proof. $\forall i \in \{(j+1), \ldots, k\}$ $V((v_1, \ldots, v_{i-1}, *, \ldots, *))$ $= V((v_1, \ldots, v_{i-1}, v_i, *, \ldots, *)) \cup$ $V((v_1, \ldots, v_{i-1}, \overline{v}_i, *, \ldots, *))$ □

The following corollary is a direct consequence of Lemmas 6.1 and 6.2,

Corollary 6.3. *The following cases state the conditions to ensure that an s–face $s = (v_1, \ldots, v_j, *, \ldots, *)$ contains at least one of the vertices of the hyperplane–hypercube intersection:*

1. ***When** $j = n$. The subset of nodes is a single vertex. A vertex $v = (v_1, \ldots, v_n)$ is a solution if $\sum_{j=1}^{n} v_j = \alpha$, otherwise at least one solution is at its edges iff $\sum_{j=1}^{n} v_j < \alpha$ and $\exists v_i \in \{v_1, \ldots, v_n\}$ $(v_i = 0) \wedge \left(b_i > \alpha - \sum_{j=1}^{n} v_j\right)$. Namely, the solution is at $(v_1, \ldots, v_{i-1}, \alpha - \sum v_j, v_{i+1}, \ldots, v_n)$.*
2. ***When** $j < n$ **and Equation 5 holds**. If $\alpha = \sum_{i=1}^{j} v_i$ the Border Vertex is the unique solution, otherwise the solutions can be found by exploring adjacent edges of the Border Vertex of s.*
3. ***When** $j < n$ **and Equation 5 does not hold**. Solutions exists if $\sum_{i=1}^{j} v_i < \alpha$ and $\sum_{i=1}^{j} v_i + \sum_{i=j+1}^{n} b_i \geq \alpha$. To find solutions, a partition of the subset of nodes using Lemma 6.2 must be done to find subsets of nodes that match with one of the previous cases.*

Given a hyperbox B and a hyperplane defined by α, Algorithm 2 determines all vertices of their intersection. At Step 1, $\mathcal{L}$ and $\mathcal{F}$ are initialized; $\mathcal{L}$ is a

Algorithm 2 EXPLOREBN(B, α, s)

Input: An ordered hyperbox B located at the origin with coordinates of the upper corner $(b_1, \ldots, b_n)$, and the hyperplane defined by α.
Output: Solutions for the intersection of B and α adjacent or at the vertices $V(s)$

1: $\mathcal{L} \leftarrow \{\}$, $\mathcal{F} \leftarrow \{(*, \ldots, *)\}$
2: **while** $\mathcal{F} \neq \{\}$ **do**
3: $v \leftarrow \text{pop}(\mathcal{F})$
4: $j \leftarrow$ number of defined variables of v
5: $w \leftarrow (v_1, \ldots, v_j, 0, \ldots, 0)$
6: Compute maximum $k \leq n$ **s.t.** $\sum_{i=1}^{j} v_i + \sum_{i=j+1}^{k} b_i \leq \alpha$
7: $\forall i \in \{(j+1), \ldots, k\}\, w_i \leftarrow b_i$
8: $\mathcal{L} \leftarrow \mathcal{L} \cup$ FINDSOLUTIONS(w, B, α)
9: **if** $k < n$ **then** ▷ Partition is needed
10: $v' \leftarrow (v_1, \ldots, v_j, b_{j+1}, \ldots, b_k, *, \ldots, *)$
11: $Z \leftarrow \mathcal{P}(v, v')$ ▷ Lemma 6.2
12: **for all** $z \in Z \setminus \{v'\}$ $\text{push}(\mathcal{F}, z)$
13: **end if**
14: **end while**
15: **return** $\mathcal{L}$

list to save the hyperbox-hyperplane intersection vertices, while $\mathcal{F}$ is a stack that indicates which s–faces need to be explored. After the initialization, the value $k \leq n$ is calculated at Line 6; this value is the number of variables that can be set to the upper limit without exceeding the α value. When $k = n$, a single vertex was found, and when $k < n$, an s–face with the border node $w = (v_1, \ldots, v_j, b_{j+1}, \ldots, b_k, 0, \ldots, 0)$ was found. In both cases, Line 8 computes the solutions by calling the procedure FINDSOLUTIONS.

Algorithm 3 FINDSOLUTIONS (v, B, α)

Input: An ordered hyperbox B located at the origin with coordinates of the upper corner $(b_1, \ldots, b_n)$, the hyperplane defined by α; and a vertex $v = (v_1, \ldots, v_n)$.
Output: Solutions adjacent to or at v

1: $\mathcal{L} \leftarrow \{\}$
2: $\alpha' \leftarrow \alpha - \sum_{i=1}^{n} v_i$
3: **if** $\alpha' = 0$ **then** ▷ v is a solution vertex
4: $\mathcal{L} \leftarrow \{v\}$
5: **else**
6: **for all** $i \in \{1, \ldots, n \mid b_i > \alpha', v_i = 0\}$ **do**
7: $t \leftarrow v$, $t_i \leftarrow \alpha'$, $\mathcal{L} \leftarrow \mathcal{L} \cup \{t\}$
8: **end for**
9: **end if**
10: **return** $\mathcal{L}$

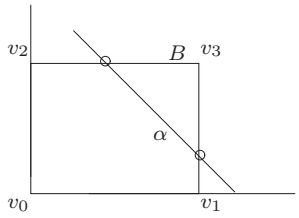


Figure 3: The intersection of a rectangle B and a line α. It is easier to find the intersection vertices starting from v_3 than starting from v_0.

In order to find the complete set of solutions for the case that $k < n$, first v is partitioned based on the s–face that contains the border node (Line 11) and then the unexplored parts of $\mathcal{P}(v, v')$ are pushed onto Stack $\mathcal{F}$ (Line 12). Algorithm 3 is an auxiliary procedure to compute all solutions adjacent to or at vertex v.

6.1 Improving EXPLOREBN

Using low border vertices is not always the best option, Figure 3 illustrates this situation. The two intersections marked with a circle can be found from the low border vertices v_1 and v_2 or from the single high border vertex v_3. The complexity of the algorithm is reduced when more solutions are found from less border vertices, it means that for some instance problems it is better to use high border vertices. Instead of reaching solutions by subtracting quantities iteratively from the upper corner, next lemma describes how to convert an instance problem in another equivalent one.

Lemma 6.4. *Let $c = (c_1, \ldots, c_n)$ be a given solution of the intersection between the hyperbox B and hyperplane α, its complement $c' = (c'_1, \ldots, c'_n)$ where $c'_i = b_i - c_i$ is a solution of the intersection between the hyperbox B and hyperplane with parameter $\alpha' = \sum_{i=1}^{n} b_i - \alpha$.*

Proof. It is important to note that if there exists a solution of B and α then $0 \leq \alpha \leq \sum_{i=0}^{n} b_i$. A solution $c = (c_1, \ldots, c_n)$ must live in some edge, then $c' = (c'_1, \ldots, c'_n)$ also lives in some edge because exists a unique coordinate c_i such that $c_i, c'_i \in [\bar{b}_i, b_i]$, and $\forall j \neq i$ $c_j, c'_j \in \{\bar{b}_j, b_j\}$. As $\sum_{i=1}^{n} c_i = \alpha$ then $\sum_{i=1}^{n} c'_i = \sum_{i=1}^{n} (b_i - c_i) = \sum_{i=1}^{n} b_i - \sum_{i=1}^{n} c_i = \sum_{i=1}^{n} b_i - \alpha = \alpha'$ □

Using Lemma 6.4 one can find the solutions for the intersection of B and α by solving the intersection of B and the hyperplane defined by $\alpha' = \sum_{i=1}^{n} b_i - \alpha$ and then mapping the solutions c' to c. To choose between solving the direct or the indirect problem, Figure

3 illustrates that one can reach more solutions from border nodes closer to $(0,\ldots,0)$ or $(b_1,\ldots,b_n)$. Based on the k value calculated at Line 6 of Algorithm 2, we decide to solve the direct or the indirect problem. As a rule of thumb is better to solve the direct problem when $k \leq n-k$.

6.2 Computational Complexity

To determine the time complexity of the proposed algorithm, we use Lemma 6.5

Lemma 6.5. *Let $s = (v_1,\ldots,v_j,*,\ldots,*)$, $s' = (v_1,\ldots,v_j,w_{j+1},\ldots,w_k,*,\ldots,*)$ be two s–faces such that $k < n$ and s' has a single border node, then each element of $\mathcal{P}(s,s')$ has at least one solution.*

Proof. If $k < n$ then s' has at least one undefined variable, $\forall i \in \{(j+1),\ldots,k\}$ an element of $\mathcal{P}(s,s')$ is obtained by placing $\overline{w}_i$ at the i–th position of s'. Because s' has one or more solutions, and for an ordered hyperbox $\overline{w}_i \leq b_n$, then each element of $\mathcal{P}(s,s')$ has at least one solution. □

A direct consequence of Lemma 6.5 is that every s-face pushed into the stack $\mathcal{F}$ (Algorithm 2, Line 12) produces at least one solution. Steps 3 to 13 of Algorithm 2, including the execution of FINDSOLUTIONS, cost $O(n)$. Given that each iterative step finds one or more solutions, the while loop will be executed at most m times, where m is the number of solutions. Then the complexity for the proposed algorithm is $O(nm)$.

6.3 Extensions

The basic method presented in Algorithm 2 seems to be limited to an ordered hyperbox and a hyperplane in the form of Equation 2. However, this shortcomings can be overcome by straightforward extensions to the basic method. These extensions, include:

Scaling A problem with a hyperbox $B = [0,b_1]\times\ldots\times[0,b_n]$ and a hyperplane in the form $\sum_{i=1}^{n}\beta_i x_i = \alpha$ s.t. $\forall i \in \{1,\ldots,n\}\ \beta_i \neq 0$ can be mapped to one of the form $\sum_{i=n}^{n} x_i = \alpha$ with $B^\star = [0,\beta_1 b_1]\times\ldots\times[0,\beta_n b_n]$. Every solution $c^\star = (c_1^\star,\ldots,c_n^\star)$ maps to the solution $c = \left(\frac{c_1^\star}{\beta_1},\ldots,\frac{c_n^\star}{\beta_n}\right)$ in the original problem.

Translation A problem with a hyperbox $B = [a_1,b_1]\times\ldots\times[a_n,b_n]$ and a hyperplane in the form $\sum_{i=1}^{n} x_i = \alpha$ can be mapped to one of the form $\sum_{i=n}^{n} x_i^\star = \alpha - \sum_{i=1}^{n} a_i$ with $B^\star = [0,b_1-a_1]\times\ldots\times[0,b_n-a_n]$. Every solution $c^\star = (c_1^\star,\ldots,c_n^\star)$ maps to the solution $c = (c_1^\star + a_1,\ldots,c_n^\star + a_n)$ in the original problem.

These pre- and post-processing mappings allow us to solve the general problem, concerning the intersection of any hyperbox with any hyperplane, without changing the overall time complexity.

7 Experimental Results

In order to evaluate the proposed algorithm, we create random instance problems. In the first test we create a hyperbox of dimension $n = 20$ (by considering each upper limit as an independent random variable with a uniform distribution), then we increment iteratively the parameter α from 0 to $\sum_{i=1}^{n} b_i$. For every plane the intersection points were found. Figure 4 shows the results for this test. The computational cost for the naive algorithm is constant, since all the $n2^{n-1}$ edges must be explored to find the (potential) intersection points. The proposed algorithm exhibits a linearly bounded behavior with respect to the number of solutions: this is a consequence of the fact that only the strictly necessary nodes were explored.

The second test consists of finding the solutions by varying the dimension of the problem. Again for each dimension we generate a random hyperbox B. For each hyperbox the alpha value was incremented from 0 to $\sum_{i=1}^{n} b_i$. Figure 5(a) shows the results for this test without the improvement proposed in Section 6.1; the elapsed time depends on the number of solutions for each border node. The elapsed time also depends on the median distance of the border nodes to $(0,\ldots,0)$; then there are two curves for each dimension. Upper curve grows above linear because the α plane for these instances is closer to the upper node of the hyperbox, therefore, we find less solutions per border node. Figure 5(b) is the same experiment with the improvement implemented. That is, when the α plane is closer to the upper node of the hyperbox, the search starts from there, instead of

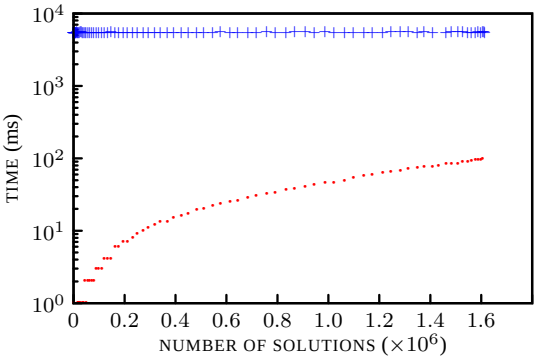
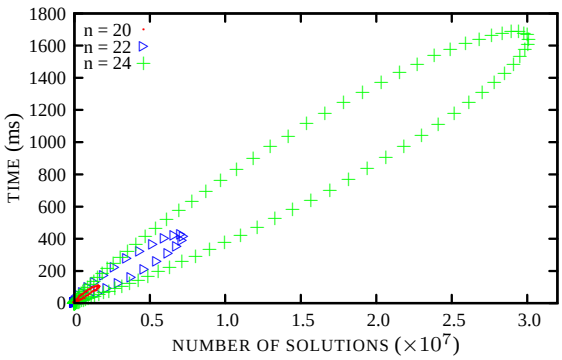
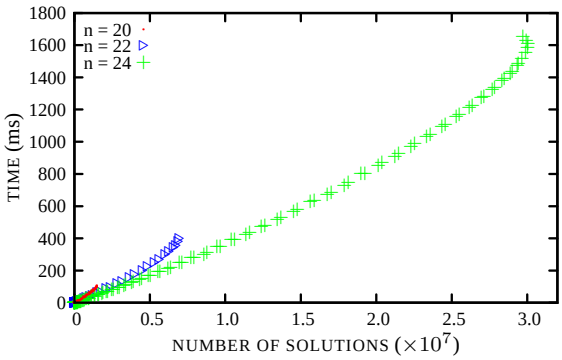


Figure 4: Naive algorithm (crosses) against EXPLOREBN algorithm (dots) for $n = 20$.



(a) Initial version



(b) Using improvement proposed in Section 6.1

Figure 5: Behavior of the EXPLOREBN algorithm for different dimensions.

starting from the origin. This improvement allows us to compute the intersection in $O(mn)$ time in the worst case, yielding cheaper computational cost.

8 Conclusions

This paper presents an algorithm to compute the intersection between a hyperbox and a hyperplane. This problem arises from many optimization problems where the hyperbox represents the operation region (search space) and the hyperplane represents an equality constraint.

The difference in time achieved by Algorithm EXPLOREBN was considerable with respect to the naive algorithm – see Figure 4. The algorithm exhibits a time complexity of $O(mn)$, where m is the number of solutions and n the problem's dimensionality. A further improvement allows the algorithm to achieve a time complexity lower than that.

The algorithm was implemented in Java and Mathematica and can be found at `http://lsc.fie.umich.mx/~juan/hypercubes`.

References

[1] Calderon, F., Fuerte-Esquivel, C. R., Flores, J. J., and Silva, J. C. A constraint-handling genetic algorithm to power economic dispatch. In *MICAI '08: Proceedings of the 7th Mexican International Conference on Artificial Intelligence*, pages 371–381, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] Garbano, M., Malerba, J., and Lewinter, M. Hypercubes and pascal's triangle: A tale of two proofs. *Math. Mag*, pages 216–217, 2003.

[3] Graham, R. and Pollak, H. On embedding graphs in squashed cubes. *Springer Lecture Notes Math*, 303:99–110, 1972.

[4] Harary, F., Hayes, J. P., and Wu, H.-J. A survey of the theory of hypercube graphs. *Comput. Math. Applic.*, 15(4):277–289, 1988.

[5] Klavzăr, S. Counting hypercubes in hypercubes. *Discrete Mathematics*, 306(22):2964–2967, 2006.

[6] Mount, D. M. Geometric intersection. In Goodman, J. and J. O'Rourke, editors, *The Handbook of Discrete and Computational Geometry*. CRC Press LLC, Boca Raton, FL, pages 615–630, 1997.

[7] Rezk-Salama, C. and Kolb, A. A Vertex Program for Efficient Box-Plane Intersection. In *Proc. Vision, Modeling and Visualization (VMV)*, pages 115–122, 2005.

[8] Sýkora, O. and Vrto, I. On the crossing number of hypercubes and cube connected cycles. In Schmidt, G. and Berghammer, R., editors, *Proc. 17th Intl. Workshop on Graph Theoretic Concepts in Computer Science WG'91*, LNCS 570, pages 214–218, Berlin, 1992. Springer-Verlag.

[9] Wang, R. L. and Okazaki, K. Solving the minimum crossing number problem using an improved artificial neural network. In *ICMLC*, pages 797–803, 2005.

[10] Wolsey, L. A. and Nemhauser, G. L. *Integer and Combinatorial Optimization*. 1st ed. Wiley-Interscience, November 1999.