An Efficient Implementation of a Joint Generation Algorithm*

E. Boros[1], K. Elbassioni[1], V. Gurvich[1], and L. Khachiyan[2]

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA.
{boros,elbassio,gurvich}@rutcor.rutgers.edu
[2] Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8003, USA.
leonid@cs.rutgers.edu

Abstract. Let $\mathcal{C}$ be an n-dimensional integral box, and π be a monotone property defined over the elements of $\mathcal{C}$. We consider the problems of incrementally generating jointly the families $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ of all minimal subsets satisfying property π and all maximal subsets not satisfying property π, when π is given by a polynomial-time satisfiability oracle. Problems of this type arise in many practical applications. It is known that the above joint generation problem can be solved in incremental quasi-polynomial time. In this paper, we present an efficient implementation of this procedure. We present experimental results to evaluate our implementation for a number of interesting monotone properties π.

1 Introduction

Let $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$ be an integral box, where $\mathcal{C}_1, \ldots, \mathcal{C}_n$ are finite sets of integers. For a subset $\mathcal{A} \subseteq \mathcal{C}$, let us denote by $\mathcal{A}^+ = \{x \in \mathcal{C} \mid x \geq a, \text{ for some } a \in \mathcal{A}\}$ and $\mathcal{A}^- = \{x \in \mathcal{C} \mid x \leq a, \text{ for some } a \in \mathcal{A}\}$, the ideal and filter generated by $\mathcal{A}$. Any element in $\mathcal{C} \setminus \mathcal{A}^+$ is called *independent of* $\mathcal{A}$, and we let $\mathcal{I}(\mathcal{A})$ denote the set of all *maximal independent* elements for $\mathcal{A}$. Call a family of vectors $\mathcal{A}$ *Sperner* if $\mathcal{A}$ is an antichain, i.e. if no two elements are comparable in $\mathcal{A}$. If $\mathcal{C}$ is the Boolean cube $2^{[n]}$, we get the well-known definitions of a hypergraph $\mathcal{A}$ and its family of maximal independent sets $\mathcal{I}(\mathcal{A})$.

Let $\pi : \mathcal{C} \mapsto \{0, 1\}$ be a *monotone* property defined over the elements of $\mathcal{C}$: if $x \in \mathcal{C}$ satisfies property π, i.e. $\pi(x) = 1$, then any $y \in \mathcal{C}$ such that $y \geq x$ also satisfies π. We assume that π is described by a polynomial satisfiability oracle $\mathcal{O}_\pi$, i.e. an algorithm that can decide whether a given vector $x \in \mathcal{C}$ satisfies π, in time polynomial in n and the size $|\pi|$ of the input description of π. Denote

* This research was supported by the National Science Foundation (Grant IIS-0118635). The third author is also grateful for the partial support by DIMACS, the National Science Foundation's Center for Discrete Mathematics and Theoretical Computer Science.

C.C. Ribeiro and S.L. Martins (Eds.): WEA 2004, LNCS 3059, pp. 114–128, 2004.

respectively by $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ the families of minimal elements satisfying property π, and maximal elements not satisfying property π. Then it is clear that $\mathcal{G}_\pi = \mathcal{I}(\mathcal{F}_\pi)$ for any monotone property π. Given a monotone property π, we consider the problem of jointly generating the families $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$:

GEN$(\mathcal{C}, \mathcal{F}_\pi, \mathcal{G}_\pi, \mathcal{X}, \mathcal{Y})$**:** *Given a monotone property π, represented by a satisfiability oracle $\mathcal{O}_\pi$, and two explicitly listed vector families $\mathcal{X} \subseteq \mathcal{F}_\pi \subseteq \mathcal{C}$ and $\mathcal{Y} \subseteq \mathcal{G}_\pi \subseteq \mathcal{C}$, either find a new element in $(\mathcal{F}_\pi \setminus \mathcal{X}) \cup (\mathcal{G}_\pi \setminus \mathcal{Y})$, or prove that these families are complete: $(\mathcal{X}, \mathcal{Y}) = (\mathcal{F}_\pi, \mathcal{G}_\pi)$.*

It is clear that for a given monotone property π, described by a satisfiability oracle $\mathcal{O}_\pi$, we can generate both $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ simultaneously by starting with $\mathcal{X} = \mathcal{Y} = \emptyset$ and solving problem GEN$(\mathcal{C}, \mathcal{F}_\pi, \mathcal{G}_\pi, \mathcal{X}, \mathcal{Y})$ for a total of $|\mathcal{F}_\pi|+|\mathcal{G}_\pi|+1$ times, incrementing in each iteration either $\mathcal{X}$ or $\mathcal{Y}$ by the newly found vector $x \in (\mathcal{F}_\pi \setminus \mathcal{X}) \cup (\mathcal{G}_\pi \setminus \mathcal{Y})$, according to the answer of the oracle $\mathcal{O}_\pi$, until we have $(\mathcal{X}, \mathcal{Y}) = (\mathcal{F}_\pi, \mathcal{G}_\pi)$.

In most practical applications, the requirement is to generate either the family $\mathcal{F}_\pi$ or the family $\mathcal{G}_\pi$, i.e. we consider the separate generation problems:

GEN$(\mathcal{C}, \mathcal{F}_\pi, \mathcal{X})$ **(*GEN***$(\mathcal{C}, \mathcal{G}_\pi, \mathcal{Y})$**):** *Given a monotone property π and a subfamily $\mathcal{X} \subseteq \mathcal{F}_\pi \subseteq \mathcal{C}$ (respectively, $\mathcal{Y} \subseteq \mathcal{G}_\pi \subseteq \mathcal{C}$), either find a new minimal satisfying vector $x \in \mathcal{F}_\pi \setminus \mathcal{X}$ (respectively, maximal non-satisfying vector $x \in \mathcal{G}_\pi \setminus \mathcal{Y}$), or prove that the given partial list is complete: $\mathcal{X} = \mathcal{F}_\pi$ (respectively, $\mathcal{Y} = \mathcal{G}_\pi$).*

Problems GEN$(\mathcal{C}, \mathcal{F}_\pi, \mathcal{X})$ and GEN$(\mathcal{C}, \mathcal{G}_\pi, \mathcal{Y})$ arise in many practical applications and in a variety of fields, including artificial intelligence [14], game theory [18,19], reliability theory [8,12], database theory [9,14,17], integer programming [4,6,20], learning theory [1], and data mining [2,6,9]. Even though these two problems may be NP-hard in general (see e.g. [20]), it is known that the joint generation problem GEN$(\mathcal{C}, \mathcal{F}_\pi, \mathcal{G}_\pi, \mathcal{X}, \mathcal{Y})$ can be solved in incremental quasi-polynomial time $poly(n, \log \|\mathcal{C}\|_\infty) + m^{\text{polylog } m}$ for any monotone property π described by a satisfiability oracle, where $\|\mathcal{C}\|_\infty = \sum_{i=1}^{n} |\mathcal{C}_i|$ and $m = |\mathcal{X}| + |\mathcal{Y}|$, see [4,15]. In particular, there is a polynomial-time reduction from the joint generation problem to the following problem, known as dualization on boxes, see [4, 10,16]:

DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$**:** *Given a family of vectors $\mathcal{A} \subseteq \mathcal{C}$, and a subset $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ of its maximal independent vectors, either find a new maximal independent vector $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$, or prove that no such vector exists, i.e., $\mathcal{B} = \mathcal{I}(\mathcal{A})$.*

The currently best known algorithm for dualization runs in time $poly(n) + m^{o(\log m)}$, see [4,15]. Unfortunately, this joint generation may not be an efficient algorithm for solving either of GEN$(\mathcal{C}, \mathcal{F}_\pi, \mathcal{X})$ or GEN$(\mathcal{C}, \mathcal{G}_\pi, \mathcal{Y})$ separately for the simple reason that we do not control which of the families $\mathcal{F}_\pi \setminus \mathcal{X}$ and $\mathcal{G}_\pi \setminus \mathcal{Y}$ contains each new vector produced by the algorithm. Suppose we want to generate $\mathcal{F}_\pi$ and the family $\mathcal{G}_\pi$ is exponentially larger than $\mathcal{F}_\pi$. Then, if we

are unlucky, we may get elements of $\mathcal{F}_\pi$ with exponential delay, while getting large subfamilies of $\mathcal{G}_\pi$ (which are not needed at all) in between. However, there are two reasons that we are interested in solving the joint generation problem efficiently. First, it is easy to see [17] that no satisfiability oracle based algorithm can generate $\mathcal{F}_\pi$ in fewer than $|\mathcal{F}_\pi| + |\mathcal{G}_\pi|$ steps, in general:

Proposition 1. *Consider an arbitrary algorithm A, which generates the family $\mathcal{F}_\pi$ by using only a satisfiability oracle $\mathcal{O}_\pi$ for the monotone property π. Then, for the algorithm to generate the whole family $\mathcal{F}_\pi$, it must call the oracle at least $|\mathcal{F}_\pi| + |\mathcal{G}_\pi|$ times.*

Proof. Clearly, any monotone property π can be defined by its value on the boundary $\mathcal{F}_\pi \cup \mathcal{G}_\pi$. For any $y \in \mathcal{F}_\pi \cup \mathcal{G}_\pi$, let us thus define the boundary of the monotone property π_y as follows:

$$\pi_y(x) = \begin{cases} \pi(x) & \text{if } x \neq y \\ \overline{\pi}(x) & \text{if } x = y. \end{cases}$$

Then π_y is a monotone property different from π, and algorithm A must be able to distinguish the Sperner families described by π and π_y for every $y \in \mathcal{F}_\pi \cup \mathcal{G}_\pi$. □

Second, for a wide class of Sperner families (or equivalently, monotone properties), the so-called *uniformly dual-bounded* families, it was realized that the size of the dual family $\mathcal{I}(\mathcal{F}_\pi)$ is uniformly bounded by a (quasi-)polynomial in the size of $\mathcal{F}_\pi$ and the oracle description:

$$|\mathcal{I}(\mathcal{X}) \cap \mathcal{I}(\mathcal{F}_\pi)| \leq (quasi\text{-})poly(|\mathcal{X}|, |\pi|) \tag{1}$$

for any non-empty subfamily $\mathcal{X} \subseteq \mathcal{F}_\pi$. An inequality of the form (1) would imply that joint generation is an incrementally efficient way for generating the family $\mathcal{F}_\pi$ (see Section 2 below and also [5] for several interesting examples).

In [7], we presented an efficient implementation of a quasi-polynomial dualization algorithm on boxes. Direct application of this implementation to solve the joint generation problem may not be very efficient since each call to the dualization code requires the construction of a new recursion tree, wasting therefore information that might have been collected from previous calls. A much more efficient approach, which we implement in this paper, is to use the same recursion tree for generating all elements of the families $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$. The details of this method will be described in Sections 3 and 4. In Section 2, we give three examples of monotone properties, that will be used in our experimental study. Finally, Section 5 presents our experimental findings, and Section 6 provides some conclusions.

2 Examples of Monotone Properties

We consider the following three monotone properties in this paper:

Monotone systems of linear inequalities. Let $A \in \mathbb{R}^{r\times n}$ be a given non-negative real matrix, $b \in \mathbb{R}^r$ be a given r-vector, $c \in \mathbb{R}^n_+$ be a given non-negative n-vector, and consider the system of linear inequalities:

$$Ax \geq b, \quad x \in \mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}. \tag{2}$$

For $x \in \mathcal{C}$, let $\pi_1(x)$ be the property that x satisfies (2). Then the families $\mathcal{F}_{\pi_1}$ and $\mathcal{G}_{\pi_1}$ correspond respectively to the minimal feasible and maximal infeasible vectors for (2). It is known [4] that the family $\mathcal{F}_{\pi_1}$ is (uniformly) dual bounded:

$$|\mathcal{I}(\mathcal{F}_{\pi_1})| \leq rn|\mathcal{F}_{\pi_1}|. \tag{3}$$

Minimal infrequent and maximal frequent sets in binary databases. Let $\mathcal{D} : R\times V \mapsto \{0,1\}$ be a given $r\times n$ binary matrix representing a set R of transactions over a set of attributes V. To each subset of columns $X \subseteq V$, let us associate the subset $S(X) = S_{\mathcal{D}}(X) \subseteq R$ of all those rows $i \in R$ for which $\mathcal{D}(i,j) = 1$ in every column $j \in X$. The cardinality of $S(X)$ is called the *support* of X. Given an integer t, a column set $X \subseteq V$ is called *t-frequent* if $|S(X)| \geq t$ and otherwise, is said to be *t-infrequent*. For each set $X \in \mathcal{C} \stackrel{\text{def}}{=} 2^V$, let $\pi_2(X)$ be the property that X is t-infrequent. Then π_2 is a monotone property and the families $\mathcal{F}_{\pi_2}$ and $\mathcal{G}_{\pi_2}$ correspond respectively to minimal infrequent and maximal frequent sets for $\mathcal{D}$. It is known [9] that

$$|\mathcal{I}(\mathcal{F}_{\pi_2})| \leq (r-t+1)|\mathcal{F}_{\pi_2}|. \tag{4}$$

Problems GEN$(\mathcal{C}, \mathcal{F}_{\pi_2}, \mathcal{Y})$ and GEN$(\mathcal{C}, \mathcal{G}_{\pi_2}, \mathcal{Y})$ appear in data mining applications, see e.g. [17].

Sparse boxes for multi-dimensional data. Let $\mathcal{S}$ be a set of points in $\mathbb{R}^n$, and $t \leq |\mathcal{S}|$ be a given integer. A *maximal t-box* is a closed n-dimensional interval which contains at most t points of $\mathcal{S}$ in its interior, and which is maximal with respect to this property (i.e., cannot be extended in any direction without strictly enclosing more points of $\mathcal{S}$). Define $\mathcal{C}_i = \{p_i \mid p \in \mathcal{S}\}$ for $i = 1,\ldots,n$ and consider the family of boxes $\mathcal{B} = \{[a,b] \subseteq \mathbb{R}^n \mid a,b \in \mathcal{C}_1 \times \cdots \times \mathcal{C}_n,\ a \leq b\}$. For $i = 1,\ldots,n$, let $u_i = \max \mathcal{C}_i$, and let $\mathcal{C}_{i+n} \stackrel{\text{def}}{=} \{u_i - p \mid p \in \mathcal{C}_i\}$ be the chain ordered in the direction opposite to $\mathcal{C}_i$. Consider the $2n$-dimensional box $\mathcal{C} = \mathcal{C}_1\times\cdots\times\mathcal{C}_n\times\mathcal{C}_{n+1}\times\cdots\times\mathcal{C}_{2n}$ and let us represent every n-dimensional interval $[a,b] \in \mathcal{B}$ as the $2n$-dimensional vector $(a, u-b) \in \mathcal{C}$, where $u = (u_1,\ldots,u_n)$. This gives a monotone injective mapping $\mathcal{B} \mapsto \mathcal{C}$ (not all elements of $\mathcal{C}$ define a box, since $a_i > b_i$ is possible for $(a, u-b) \in \mathcal{C}$). Let us now define the monotone property π_3 to be satisfied by an $x \in \mathcal{C}$ if and only if x does not define a box, or the box defined by x contains at most t points of S in its interior. Then the sets

$\mathcal{F}_{\pi_3}$ and $\mathcal{G}_{\pi_3}$ can be identified respectively with the set of maximal t-boxes (plus a polynomial number of non-boxes), and the set of minimal boxes of $x \in \mathcal{B} \subseteq \mathcal{C}$ which contain at least $k+1$ points of $\mathcal{S}$ in their interior. It is known [6] that the family of maximal t-boxes is (uniformly) dual-bounded:

$$|\mathcal{I}(\mathcal{F}_{\pi_3})| \leq |\mathcal{S}||\mathcal{F}_{\pi_3}|. \tag{5}$$

The problem of generating all elements of $\mathcal{F}_{\pi_3}$ has been studied in the machine learning and computational geometry literatures (see [11,13,23]), and is motivated by the discovery of missing associations or "holes" in data mining applications (see [3,21,22]). [13] gives an algorithm, for solving this problem, whose worst-case time complexity is exponential in the dimension n of the given point set.

3 Terminology and Outline of the Algorithm

Throughout the paper, we assume that we are given an integer box $\mathcal{C}^* = \mathcal{C}_1^* \times \ldots \times \mathcal{C}_n^*$, where $\mathcal{C}_i^* = [l_i^* : u_i^*]$, and $l_i^* \leq u_i^*$, are integers, and a monotone property π, described by a polynomial time satisfiability oracle, for which it is required to generate the families $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$. The generation algorithm, considered in this paper, is based on a reduction to a dualization algorithm of [4], for which an efficient implementation was given in [7]. For completeness, we briefly outline this algorithm. The problem is solved by decomposing it into a number of smaller subproblems and solving each of them recursively. The input to each such subproblem is a sub-box $\mathcal{C}$ of the original box $\mathcal{C}^*$ and two subsets $\mathcal{F} \subseteq \mathcal{F}^*$ and $\mathcal{G} \subseteq \mathcal{G}^*$ of integral vectors, where $\mathcal{F}^* \subseteq \mathcal{F}_\pi$ and $\mathcal{G}^* \subseteq \mathcal{G}_\pi$ denote respectively the subfamilies of minimal satisfying and maximal non-satisfying vectors that have been generated so far. Note that, by definition, the following condition holds for the original problem and all subsequent subproblems:

$$a \not\leq b, \quad \text{for all } a \in \mathcal{F}, b \in \mathcal{G}. \tag{6}$$

Given an element $a \in \mathcal{F}$ ($b \in \mathcal{G}$), we say that a coordinate $i \in [n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$ is *essential* for a (respectively, b), in the box $\mathcal{C} = [l_1 : u_1] \times \cdots \times [l_n : u_n]$, if $a_i > l_i$ (respectively, if $b_i < u_i$). Let us denote by $\mathrm{Ess}(x)$ the set of essential coordinates of an element $x \in \mathcal{F} \cup \mathcal{G}$. Finally, given a sub-box $\mathcal{C} \subseteq \mathcal{C}^*$, and two subsets $\mathcal{F} \subseteq \mathcal{F}^*$ and $\mathcal{G} \subseteq \mathcal{G}^*$, we shall say that $\mathcal{F}$ is *dual to* $\mathcal{G}$ in $\mathcal{C}$ if $\mathcal{F}^+ \cup \mathcal{G}^- \supseteq \mathcal{C}$.

A key lemma, on which the algorithm in [4] is based, is that either (i) there is an element $x \in \mathcal{F} \cup \mathcal{G}$ with at most $1/\epsilon$ essential coordinates, where $\epsilon \stackrel{\text{def}}{=} 1/(1 + \log m)$ and $m \stackrel{\text{def}}{=} |\mathcal{F}| + |\mathcal{G}|$, or (ii) one can easily find a new element $z \in \mathcal{C} \setminus (\mathcal{F}^+ \cup \mathcal{G}^-)$, by picking each element z_i independently at random from $\{l_i, u_i\}$ for $i = 1, \ldots, n$; see subroutine Random solution() in the next section. In case (i), one can decompose the problem into two strictly smaller subproblems as follows. Assume, without loss of generality, that $x \in \mathcal{F}$ has at most $1/\epsilon$ essential

coordinates. Then, by (6), there is an $i \in [n]$ such that $|\{b \in \mathcal{G} \ : \ b_i < x_i\}| \geq \epsilon|\mathcal{G}|$. This allows us to decompose the original problem into two subproblems $\mathrm{GEN}(\mathcal{C}', \pi, \mathcal{F}, \mathcal{G}')$ and $\mathrm{GEN}(\mathcal{C}'', \pi, \mathcal{F}'', \mathcal{G})$, where $\mathcal{C}' = \mathcal{C}_1 \times \cdots \times \mathcal{C}_{i-1} \times [x_i : u_i] \times \mathcal{C}_{i+1} \times \cdots \times \mathcal{C}_n$, $\mathcal{G}' = \mathcal{G} \cap \mathcal{C}^+$, $\mathcal{C}'' = \mathcal{C}_1 \times \cdots \times \mathcal{C}_{i-1} \times [l_i : x_i - 1] \times \mathcal{C}_{i+1} \times \cdots \times \mathcal{C}_n$, and $\mathcal{F}'' = \mathcal{F} \cap \mathcal{C}^-$. This way, the algorithm is guaranteed to reduce the cardinality of one of the sets $\mathcal{F}$ or $\mathcal{G}$ by a factor of at least $1 - \epsilon$ at each recursive step. For efficiency reasons, we do two modifications to this basic approach. First, we use sampling to estimate the sizes of the sets $\mathcal{G}', \mathcal{F}''$ (see subroutine Est() below). Second, once we have determined the new sub-boxes $\mathcal{C}', \mathcal{C}''$ above, we do not compute the *active* families $\mathcal{G}'$ and $\mathcal{F}''$ at each recursion step (this is called the Cleanup step in the next section). Instead, we perform the cleanup step only when the number of vectors reduces by a certain factor f, say $1/2$, for two reasons: First, this improves the running time since the elimination of vectors is done less frequently. Second, the expected total memory required by all the nodes of the path from the root of the recursion tree to a leaf is at most $O(nm + m/(1-f))$, which is linear in m for constant $f < 1$.

4 The Algorithm

We use the following data structures in our implementation:

- Two arrays of vectors, F and G containing the elements of $\mathcal{F}^*$ and $\mathcal{G}^*$ respectively.
- Two (dynamic) arrays of indices, index($\mathcal{F}$) and index($\mathcal{G}$), containing the indices of vectors from $\mathcal{F}^*$ and $\mathcal{G}^*$ (i.e. containing pointers to elements of the arrays F and G), that appear in the current subproblem. These arrays are used to enable sampling from the sets $\mathcal{F}$ and $\mathcal{G}$, and also to keep track of which vectors are currently active, i.e, intersect the current box.
- Two balanced binary search trees $\mathbf{T}(\mathcal{F}^*)$ and $\mathbf{T}(\mathcal{G}^*)$, built on the elements of $\mathcal{F}^*$ and $\mathcal{G}^*$ respectively using lexicographic ordering. Each node of the tree $\mathbf{T}(\mathcal{F}^*)$ ($\mathbf{T}(\mathcal{G}^*)$) contains an index of an element in the array F (G). This way, checking whether a given vector $x \in \mathcal{C}$ belongs to $\mathcal{F}^*$ ($\mathcal{G}^*$) or not, takes only $O(n \log |\mathcal{F}^*|)$ ($O(n \log |\mathcal{G}^*|)$) time.

In the sequel, we let $m = |\mathcal{F}| + |\mathcal{G}|$ and $\epsilon = 1/(1 + \log m)$. We use the following subroutines in our implementation:

Minimization $\min_{\mathcal{F}}(z)$**.** It takes as input a vector $z \in \mathcal{F}^+$ and returns a minimal vector z^* in $\mathcal{F}^+ \cap \{z\}^-$. Such a vector $z^* = \min_{\mathcal{F}}(z)$ can, for instance, be computed by coordinate descent:

$$
\begin{aligned}
z_1^* &\leftarrow \min\{y_1 \mid (y_1, y_2, \ldots, y_{n-1}, y_n) \in \mathcal{F}^+ \cap \{z\}^-\},\\
z_2^* &\leftarrow \min\{y_2 \mid (z_1^*, y_2, \ldots, y_{n-1}, y_n) \in \mathcal{F}^+ \cap \{z\}^-\},\\
&\ldots\\
z_n^* &\leftarrow \min\{y_n \mid (z_1^*, z_2^*, \ldots, z_{n-1}^*, y_n) \in \mathcal{F}^+ \cap \ \{z\}^-\}.
\end{aligned}
$$

Note that each of the n coordinate steps in the above procedure can be reduced via binary search to at most $\log(\|\mathcal{C}\|_\infty + 1)$ satisfiability oracle calls for the monotone property π.

More efficient procedures can be obtained if we specialize this routine to the specific monotone property under consideration. For instance, for property π_1 this operation can be performed in $O(nr)$ steps as follows. For $j = 1, \ldots, r$, let $a_j x \geq b_j$ be the jth inequality of the system. We initialize $z^* \leftarrow z$ and $w_j = a_j z^* - b_j$ for $j = 1, \ldots, r$. For the ith step of the coordinate descend operation, we let $z_i^* \leftarrow z_i^* - \lfloor \lambda \rfloor$ and $w_j \leftarrow w_j - \lfloor \lambda \rfloor a_{ji}$ for $j = 1, \ldots, r$, where

$$\lambda = \min_{1 \leq j \leq r} \left\{ \frac{w_j}{a_{ji}} \ : \ a_{ji} > 0 \right\}.$$

Now consider property π_2. Given a set $Z \in \mathcal{F}^+_{\pi_2}$, the operation $\min_{\mathcal{F}_{\pi_2}}(Z)$ can be done in $O(nr)$ by initializing $Z^* \leftarrow Z$, $s \leftarrow |S(Z)|$ and $c(Y) \leftarrow |Z \setminus Y|$ for all $Y \in \mathcal{D}$, and repeating, for $i \in Z$, the following two steps: (i) $\mathcal{Y} \leftarrow \{Y \in \mathcal{D} \ : \ c(Y) = 1, \ Y \not\ni i\}$; (ii) if $|\mathcal{Y}| + s \leq t - 1$ then 1. $Z^* \leftarrow Z^* \setminus \{i\}$, 2. $s \leftarrow s + |\mathcal{Y}|$, and 3. $c(Y) \leftarrow c(Y) - 1$ for each $Y \in \mathcal{D}$ such that $Y \not\ni i$.

For the monotone property π_3 and $z \in \mathcal{C}$, the operation $\min_{\mathcal{F}_{\pi_3}}(z)$ can be done in $O(n|\mathcal{S}|)$ as follows. For each point $p \in \mathcal{S}$, let $p' \in \mathbb{R}^{2n}$ be the point with components $p'_i = p_i$ for $i = 1, \ldots, n$, and $p'_i = u_{i-n} - p_{i-n}$ for $i = n+1, \ldots, 2n$. Initialize $s(z) \leftarrow |\{p \in \mathcal{S} \ : \ p \text{ is in the interior of } z\}|$ and $c(p) \leftarrow |\{i \in [n] : p'_i \leq z_i\}|$ for all $p \in \mathcal{S}$. Repeat, for $i = 1, \ldots, 2n$, the following steps: (i) $z_i^* \leftarrow \min\{p'_i : p \in \mathcal{S}, \ c(p) = 1, \ p'_i \leq z_i \text{ and } |\{q \in \mathcal{S} \ : \ c(q) = 1, \ p'_i < q_i \leq z_i\}| \leq t - s(z)\}$; (ii) $c(p) \leftarrow c(p) - 1$ for each $p \in \mathcal{S}$ such that $z_i^* < p'_i \leq z_i$. Note that (i) can be performed in $O(|\mathcal{S}|)$ steps assuming that we know the sorted order for the points along each coordinate.

Maximization $\max_{\mathcal{G}}(z)$**.** It computes, for a given vector $z \in \mathcal{G}^-$, a maximal vector $z^* \in \mathcal{G}^- \cap z^+$. Similar to $\min_{\mathcal{F}}(z)$, this problem can be done, in general, by coordinate descent. For $\mathcal{G}_{\pi_1}$, $\mathcal{G}_{\pi_2}$ and $\mathcal{G}_{\pi_3}$, this operation can be done in $O(nr)$, $O(nr)$, and $O(n|\mathcal{S}|)$ respectively.

Below, we denote respectively by T_{min} and T_{max} the maximum time taken by the routines $\min_{\mathcal{F}_\pi}(z)$ and $\max_{\mathcal{G}_\pi}(z)$ on any point $z \in \mathcal{C}$.

Exhaustive duality$(\mathcal{C}, \pi, \mathcal{F}, \mathcal{G})$**.** Assuming $|\mathcal{F}||\mathcal{G}| \leq 1$, check if there are no other vectors in $\mathcal{C} \setminus (\mathcal{F}^+ \cup \mathcal{G}^-)$ as follows. First, if $|\mathcal{F}| = |\mathcal{G}| = 1$ then find an $i \in [n]$ such that $a_i > b_i$, where $\mathcal{F} = \{a\}$ and $\mathcal{G} = \{b\}$: (Such a coordinate is guaranteed to exist by (6))
1. If there is a $j \neq i$ such that $b_j < u_j$ then let $z = (u_1, \ldots, u_{i-1}, b_i, u_{i+1}, \ldots, u_n)$.
2. Otherwise, if there is a $j \neq i$ such that $a_j > l_j$ then let $z = (u_1, \ldots, u_{j-1}, a_j - 1, u_{j+1}, \ldots, u_n)$.
3. If $b_i < a_i - 1$ then let $z = (u_1, \ldots, u_{i-1}, a_i - 1, u_{i+1}, \ldots, u_n)$.

In cases 1, 2 and 3, return either $\min_{\mathcal{F}_\pi}(z)$ or $\max_{\mathcal{G}_\pi}(z)$ depending on whether $\pi(z) = 1$ or $\pi(z) = 0$, respectively.
4. Otherwise return *FALSE* (meaning that $\mathcal{F}$ and $\mathcal{G}$ are dual in $\mathcal{C}$).

Second, if $|\mathcal{F}| = 0$ then check satisfiability of u. If $\pi(u) = 1$ then return $\min_{\mathcal{F}_\pi}(u)$. Else, if $\pi(u) = 0$ then let $z = \max_{\mathcal{G}_\pi}(u)$, and return either *FALSE* or z depending on whether $z \in \mathcal{G}^*$ or not (this check can be done in $O(n \log |\mathcal{G}^*|)$ using the search tree $\mathbf{T}(\mathcal{G}^*)$). Finally, if $|\mathcal{G}| = 0$ then check satisfiability of l. If $\pi(l) = 0$ then return $\max_{\mathcal{G}_\pi}(l)$. Else, if $\pi(l) = 1$ then let $z = \min_{\mathcal{F}_\pi}(l)$, and return either *FALSE* or z depending on whether $z \in \mathcal{F}^*$ or not. This step takes $O(\max\{n \log |\mathcal{F}^*| + T_{min}, n \log |\mathcal{G}^*| + T_{max}\})$ time.

Random solution$(\mathcal{C}, \pi, \mathcal{F}^*, \mathcal{G}^*)$**.** Repeat the following for $k = 1, \ldots, t_1$ times, where t_1 is a constant (say 10): Find a random point $z^k \in \mathcal{C}$, by picking each coordinate z^k_i randomly from $\{l_i, u_i\}$, $i = 1, \ldots, n$. If $\pi(z^k) = 1$ then let $(z^k)^* \leftarrow \min_{\mathcal{F}_\pi}(z^k)$, and if $(z^k)^* \notin \mathcal{F}^*$ then return $(z^k)^* \in \mathcal{F}_\pi \setminus \mathcal{F}^*$. If $\pi(z^k) = 0$ then let $(z^k)^* \leftarrow \max_{\mathcal{G}_\pi}(z^k)$, and if $(z^k)^* \notin \mathcal{G}^*$ then return $(z^k)^* \in \mathcal{G}_\pi \setminus \mathcal{G}^*$. If $\{(z^1)^*, \ldots, (z^{t_1})^*\} \subseteq \mathcal{F}^* \cup \mathcal{G}^*$ then return *FALSE*. This step takes $O(\max\{n \log |\mathcal{F}^*| + T_{min}, n \log |\mathcal{G}^*| + T_{max}\})$ time, and is used to check whether $\mathcal{F}^+ \cup \mathcal{G}^-$ covers a large portion of $\mathcal{C}$.

Count estimation. For a subset $\mathcal{X} \subseteq \mathcal{F}$ (or $\mathcal{X} \subseteq \mathcal{G}$), use sampling to estimate the number $\mathrm{Est}(\mathcal{X}, \mathcal{C})$ of elements of $\mathcal{X} \subseteq \mathcal{F}$ (or $\mathcal{X} \subseteq \mathcal{G}$) that are active with respect to the current box $\mathcal{C}$, i.e. the elements of the set $\mathcal{X}' \stackrel{\text{def}}{=} \{a \in \mathcal{X} \mid a^+ \cap \mathcal{C} \neq \emptyset\}$ ($\mathcal{X}' \stackrel{\text{def}}{=} \{b \in \mathcal{X} \mid b^- \cap \mathcal{C} \neq \emptyset\}$). This can be done as follows. For $t_2 = O(\log(|\mathcal{F}|+|\mathcal{G}|)/\epsilon)$, pick elements $x^1, \ldots, x^{t_2} \in \mathcal{F}$ at random, and let the random variable $Y = \frac{|\mathcal{F}|}{t_2} * |\{x^i \in \mathcal{X}' \ : \ i = 1, \ldots .t_2\}|$. Repeat this step independently for a total of $t_3 = O(\log(|\mathcal{F}| + |\mathcal{G}|))$ times to obtain t_3 estimates $Y^1, \ldots, Y^{t_3}$, and let $\mathrm{Est}(\mathcal{X}, \mathcal{C}) = \min\{Y^1, \ldots, Y^{t_3}\}$. This step requires $O(n \log^3 m)$ time.

Cleanup($\mathcal{F}$**,**$\mathcal{C}$**) (Cleanup(**$\mathcal{G}$**,**$\mathcal{C}$**)).** Set $\mathcal{F}' \leftarrow \{a \in \mathcal{F} \mid a^+ \cap \mathcal{C} \neq \emptyset\}$ (respectively, $\mathcal{G}' \leftarrow \{b \in \mathcal{G} \mid b^- \cap \mathcal{C} \neq \emptyset\}$), and return $\mathcal{F}'$ (respectively, $\mathcal{G}'$). This step takes $O(n|\mathcal{F}|)$ (respectively, $O(n|\mathcal{G}|)$).

Now, we describe the implementation of procedure $\mathrm{GEN}(\mathcal{C}, \pi, \mathcal{F}, \mathcal{G})$ which is called initially using $\mathcal{C} \leftarrow \mathcal{C}^*$, $\mathcal{F} \leftarrow \emptyset$ and $\mathcal{G} \leftarrow \emptyset$. At the return of this call, the families $\mathcal{F}^*$ and $\mathcal{G}^*$, which are initially empty, are extended respectively by the elements in $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$. Below we assume that $f \in (0, 1)$ is a constant, say $1/2$. The families $\mathcal{F}^o$ and $\mathcal{G}^o$ represent respectively the subfamilies of $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ that are generated at each recursion tree node.

Procedure GEN$(\mathcal{C}, \pi, \mathcal{F}, \mathcal{G})$**:**
Input: A box $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$, a monotone property π, and subsets $\mathcal{F} \subseteq \mathcal{F}_\pi$, and $\mathcal{G} \subseteq \mathcal{G}_\pi$.
Output: Subsets $\mathcal{F}^o \subseteq \mathcal{F}_\pi \setminus \mathcal{F}$ and $\mathcal{G}^o \subseteq \mathcal{G}_\pi \setminus \mathcal{G}$.

1. $\mathcal{F}^o \leftarrow \emptyset$, $\mathcal{G}^o \leftarrow \emptyset$.
2. While $|\mathcal{F}||\mathcal{G}| \leq 1$
3. $z \leftarrow$ Exhaustive duality$(\mathcal{C}, \pi, \mathcal{F}, \mathcal{G})$.
4. If $z = FALSE$ then return$(\mathcal{F}^o, \mathcal{G}^o)$.
5. else if $\pi(z) = 1$ then $\mathcal{F} \leftarrow \mathcal{F} \cup \{z\}$, $\mathcal{F}^o \leftarrow \mathcal{F}^o \cup \{z\}$, $\mathcal{F}^* \leftarrow \mathcal{F}^* \cup \{z\}$.
6. else $\mathcal{G} \leftarrow \mathcal{G} \cup \{z\}$, $\mathcal{G}^o \leftarrow \mathcal{G}^o \cup \{z\}$, $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \{z\}$.
7. end while
8. $z \leftarrow$ Random Solution$(\mathcal{C}, \pi, \mathcal{F}^*, \mathcal{G}^*)$.
9. While $(z \neq FALSE)$ do
10. if $\pi(z) = 1$ then $\mathcal{F} \leftarrow \mathcal{F} \cup \{z\}$, $\mathcal{F}^o \leftarrow \mathcal{F}^o \cup \{z\}$, $\mathcal{F}^* \leftarrow \mathcal{F}^* \cup \{z\}$.
11. else $\mathcal{G} \leftarrow \mathcal{G} \cup \{z\}$, $\mathcal{G}^o \leftarrow \mathcal{G}^o \cup \{z\}$, $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \{z\}$.
12. $z \leftarrow$ Random Solution$(\mathcal{C}, \pi, \mathcal{F}^*, \mathcal{G}^*)$.
13. end while
14. $x^* \leftarrow \text{argmin}\{|\operatorname{Ess}(y)| \ : \ y \in (\mathcal{F} \cap \mathcal{C}^-) \cup (\mathcal{G} \cap \mathcal{C}^+)\}$.
15. If $x^* \in \mathcal{F}$ then
16. $i \leftarrow \text{argmax}\{\operatorname{Est}(\{b \in \mathcal{G} \ : \ b_j < x^*_j\}, \mathcal{C}) \ : \ j \in \operatorname{Ess}(x^*)\}$.
17. $\mathcal{C}' = \mathcal{C}_1 \times \cdots \times \mathcal{C}_{i-1} \times [x^*_i : u_i] \times \mathcal{C}_{i+1} \times \cdots \times \mathcal{C}_n$.
18. If $\operatorname{Est}(\mathcal{G}, \mathcal{C}') \leq f * |\mathcal{G}|$ then
19. $\mathcal{G}' \leftarrow$ Cleanup$(\mathcal{G}, \mathcal{C}')$.
20. else
21. $\mathcal{G}' \leftarrow \mathcal{G}$.
22. $(\mathcal{F}_l, \mathcal{G}_l) \leftarrow$ GEN$(\mathcal{C}', \pi, \mathcal{F}, \mathcal{G}')$.
23. $\mathcal{F}^o \leftarrow \mathcal{F}^o \cup \mathcal{F}_l$, $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_l$, $\mathcal{F}^* \leftarrow \mathcal{F}^* \cup \mathcal{F}_l$.
24. $\mathcal{G}^o \leftarrow \mathcal{G}^o \cup \mathcal{G}_l$, $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{G}_l$, $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \mathcal{G}_l$.
25. $\mathcal{C}'' = \mathcal{C}_1 \times \cdots \times \mathcal{C}_{i-1} \times [l_i : x^*_i - 1] \times \mathcal{C}_{i+1} \times \cdots \times \mathcal{C}_n$.
26. If $\operatorname{Est}(\mathcal{F}, \mathcal{C}'') \leq f * |\mathcal{F}|$ then
27. $\mathcal{F}'' \leftarrow$ Cleanup$(\mathcal{F}, \mathcal{C}'')$.
28. else
29. $\mathcal{F}'' \leftarrow \mathcal{F}$.
30. $(\mathcal{F}_r, \mathcal{G}_r) \leftarrow$ GEN$(\mathcal{C}'', \pi, \mathcal{F}'', \mathcal{G})$.
31. $\mathcal{F}^o \leftarrow \mathcal{F}^o \cup \mathcal{F}_r$, $\mathcal{F}^* \leftarrow \mathcal{F}^* \cup \mathcal{F}_r$, $\mathcal{G}^o \leftarrow \mathcal{G}^o \cup \mathcal{G}_r$, $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \mathcal{G}_r$.
32. else
33-48. Symmetric versions for Steps 16-31 above (details omitted).
49. end if
50. Return $(\mathcal{F}^o, \mathcal{G}^o)$.

The following result, regarding the expected running time of the algorithm, is inherent from [7].

Proposition 2. *The expected number of recursive calls until a new element in* $(\mathcal{F}_\pi \setminus \mathcal{F}^*) \cup (\mathcal{G}_\pi \setminus \mathcal{G}^*)$ *is output, or procedure GEN*$(\mathcal{C}, \pi, \mathcal{F}, \mathcal{G})$ *terminates is* $nm^{O(log^2 m)}$.

However, as we shall see from the experiments, the algorithm seems to practically behave much more efficiently than indicated by Proposition 2. In fact, in most of the experiments we performed, we got an almost everage linear delay (in m) for generating a new point in $(\mathcal{F}_\pi \setminus \mathcal{F}^*) \cup (\mathcal{G}_\pi \setminus \mathcal{G}^*)$.

5 Experimental Results

We performed a number of experiments to evaluate our implementation on random instances of the three monotone properties described in Section 2. The experiments were performed on a *Pentium 4* processor with 2.2 GHz of speed and 512M bytes of memory. For each monotone property π, we have limited the corresponding parameters defining the property to reasonable values such that the algorithm completes generation of the sets $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ in reasonable time. Using larger values of the parameters increases the output size, resulting in large total time, although the time per output remains almost constant. For each case, the experiments were performed 5 times, and the numbers shown in the tables below represent averages.

Tables 1 and 2 show our results for linear systems with n variables and r inequalities. Each element of the constraint matrix A and the right-hand side vector b is generated at random from 1 to 15. In the tables we show the output size, the total time taken to generate the output and the average time per each output vector. The parameter c denotes the maximum value that a variable can take. The last row of the table gives the ratio of the size of $\mathcal{F}_{\pi_1}$ to the size of $\mathcal{G}_{\pi_1}$ for comparison with the worst case bound of (3). Note that this ratio is relatively close to 1, making joint generation an efficient method for generating both families $\mathcal{F}_{\pi_1}$ and $\mathcal{G}_{\pi_1}$.

Table 1. Performance of the algorithm for property π_1, where $r = 5$ and $c = 2$.

n	10		20		30		40		50	
	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$
Output size (thousands)	0.31	0.19	9.9	5.7	49.6	20.0	127.3	59.5	195.3	74.7
Total Time (sec)	4.7	4.7	297	297	1627	1625	5844	5753	10703	10700
Time/output. (msec)	13	24	27	62	29	78	40	103	50	133
Ratio $\lvert\mathcal{G}_{\pi_1}\rvert/\lvert\mathcal{F}_{\pi_1}\rvert$	0.60		0.57		0.40		0.47		0.38	

Table 2. Performance of the algorithm for property π_1, where $n = 30$ and $c = 2$.

r	5		15		25		35		45	
	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$	$\mathcal{F}_{\pi_1}$	$\mathcal{G}_{\pi_1}$
Output size (thousands)	20.4	11.6	68.6	27.8	122.7	43.3	196.6	61.7	317.5	115.5
Total Time (sec)	408	408	2244	2242	6495	6482	15857	15856	30170	30156
Time/output. (msec)	20	50	32	90	50	158	76	258	75	260
Ratio $\lvert\mathcal{G}_{\pi_1}\rvert/\lvert\mathcal{F}_{\pi_1}\rvert$	0.57		0.41		0.35		0.31		0.36	

Tables 3 and 4 show the results for minimal infrequent/maximal frequent sets. In the tables, n, r and t denote respectively the number of columns, the number of rows of the matrix, and the threshold. Each row of the matrix was generated uniformly at random. As seen from Table 3, for $t = 1, 2$, the bias between the numbers of maximal frequent sets and minimal infrequent sets, for the shown random examples, seem to be large. This makes joint generation an efficient method for generating minimal infrequent sets, but inefficient for generating maximal frequent sets for these examples. However, we observed that this bias in numbers decreases as the threshold t becomes larger. Table 4 illustrates this on a number of examples in which larger values of the threshold were used.

Table 3. Performance of the algorithm for property π_2 for threshold $t = 1, 2$.

n, r, t	20,100,1		30,100,1		40,100,1		30,100,2		30,300,2		30,500,2	
	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$
Output size (thousands)	2.9	0.08	51.7	0.1	337.1	0.1	75.4	2.5	386.7	13.5	718.3	27.1
Total Time (sec)	60	55	1520	769	22820	3413	1962	1942	13269	13214	28824	28737
Time/output. (msec)	20	690	30	7742	68	34184	28	770	33	979	40	1062
Ratio $\lvert\mathcal{G}_{\pi_2}\rvert/\lvert\mathcal{F}_{\pi_2}\rvert$	0.0280		0.0019		0.0002		0.0335		0.0350		0.0377	

Table 4. Performance of the algorithm for property π_2 for large threshold values.

n, r, t	30,300,3		30,300,5		30,300,7		30,300,9		30,1000,20		30,1000,25	
	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$	$\mathcal{F}_{\pi_2}$	$\mathcal{G}_{\pi_2}$
Output size (thousands)	403.3	73.6	362.6	134.7	269.0	100.1	199.1	74.0	491.3	145.7	398.1	114.5
Total Time (sec)	7534	7523	6511	6508	4349	4346	3031	3029	13895	13890	9896	9890
Time/output. (msec)	19	102	18	48	17	43	15	41	28	95	25	86
Ratio $\lvert\mathcal{G}_{\pi_2}\rvert/\lvert\mathcal{F}_{\pi_2}\rvert$	0.1826		0.3715		0.3719		0.3716		0.2965		0.2877	

Figures 1 and 2 show how the output rate changes for minimal feasible/maximal infeasible solutions of linear systems and for minimal infrequent/maximal frequent sets, respectively. For minimal feasible solutions, we can see that the output rate changes almost linearly as the number of outputs increases. This is not the case for the maximal infeasible solutions, where the algorithm efficiecy decreases (the generation problem for maximal infeasible solutions is NP-hard). For minimal infrequent and maximal frequent sets, Figure 2 shows that the output rate increases very slowly. This illustrates somehow that the algorithm practically behaves much better than the quasi-polynomial bound stated in Proposition 2.

Table 5 shows the results for maximal sparse/minimal non-sparse boxes with dimension n, for a set of r random points, threshold t, and upper bound c on the coordinate of each point. As in the case of frequent sets, the bias between the numbers $\mathcal{F}_{\pi_3}$ and $\mathcal{G}_{\pi_3}$ is large for $t = 0$ but seems to decrease with larger values of the threshold. In fact, the table shows two examples in which the number of minimal non-sparse boxes is larger than the number of maximal sparse boxes.

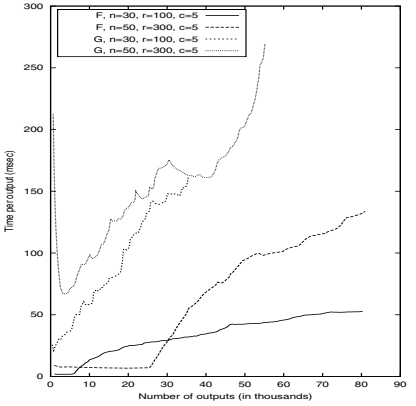

Fig. 1. Average time per output, as a function of the number of outputs for minimal feasible/maximal infeasible solutions of linear systems, with $c = 5$ and $(n, r) = (30, 100), (50, 300)$.

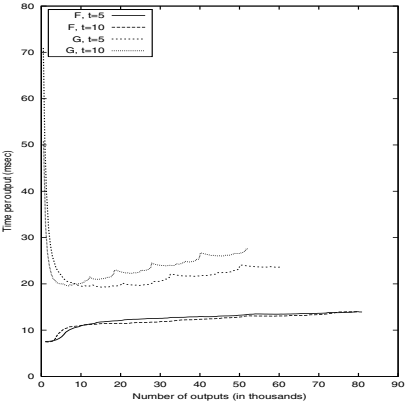

Fig. 2. Average time per output, as a function of the number of outputs for minimal infrequent/maximal frequent sets, with $n = 30$, $r = 1000$ and $t = 5, 10$.

We are not aware of any implementation of an algorithm for generating maximal sparse boxes except for [13] which presents some experiments for $n = 2$ and $t = 0$. Experiments in [13] indicated that the algorithm suggested there is almost linear in the the number of points r. Figure 3 illustrates a similar behaviour exhibited by our algorithm. In the figure, we show the total time required to generate all the 2-dimensional maximal empty boxes, as the number of points is increased from 10,000 to 60,000, for two different values of the upper bound c.

Table 5. Performance of the algorithm for property π_3 with $n = 7$ and upper bound $c = 5$.

r, t	100,0		300,0		500,0		300,2		300,6		300,10	
	$\mathcal{F}_{\pi_3}$	$\mathcal{G}_{\pi_3}$	$\mathcal{F}_{\pi_3}$	$\mathcal{G}_{\pi_3}$	$\mathcal{F}_{\pi_3}$	$\mathcal{G}_{\pi_3}$	$\mathcal{F}_{\pi_3}$	$\mathcal{G}_{\pi_3}$	$\mathcal{F}_{\pi_3}$	$\mathcal{G}_{\pi_3}$	$\mathcal{F}_{\pi_3}$	$\mathcal{G}_{\pi_3}$
Output size (thousands)	16.1	0.1	49.1	0.3	72.9	0.5	228.5	88.7	373.4	466.3	330.4	456.5
Total Time (sec)	932	623	2658	1456	3924	2933	8731	8724	17408	17404	16156	16156
Time/output. (msec)	29	6237	27	4866	27	5889	19	98	23	37	24	35
Ratio $\lvert\mathcal{G}_{\pi_3}\rvert/\lvert\mathcal{F}_{\pi_3}\rvert$	0.0062		0.0061		0.0068		0.3881		1.2488		1.3818	

As mentioned in Section 4, it is possible in general to implement the procedures $\min_{\mathcal{F}}(z)$ and $\max_{\mathcal{G}}(z)$ using the coordinate decent method, but more efficient implementations can be obtained if we specialize these procedures to the monotone property under consideration. Figure 4 compares the two different implementations for the property π_3. Clearly, the gain in performance increases as the upper bound c increases.

Let us finally point out that we have observed that the algorithm tends to run more efficiently when the sets $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ become closer in size. This observation

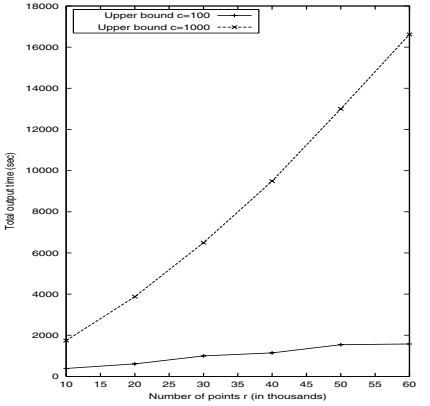

Fig. 3. Total generation time as a function of the number of points r for maximal boxes with $n = 2$, $t = 0$, and $c = 100, 1000$.

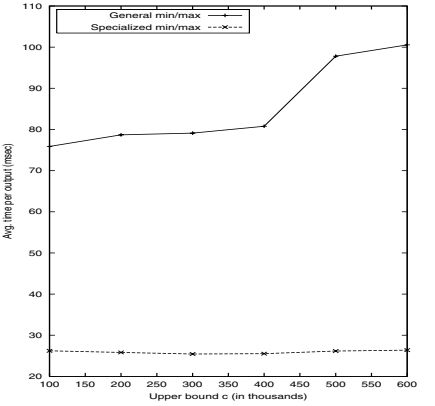

Fig. 4. Comparing general versus specialized minimization for property π_3. Each plot shows the average CPU time/maximal empty box generated versus the upper bound c, for $n = 5$ and $r = 500$.

is illustrated in Figure 5 which plots the average time per output (i.e. total time to output all the elements of $\mathcal{F}_\pi \cup \mathcal{G}_\pi$ divided by $|\mathcal{F}_\pi \cup \mathcal{G}_\pi|$) versus the ratio $|\mathcal{G}_\pi|/|\mathcal{F}_\pi|$. This indicates that, when the elements of the sets $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ are more uniformly distributed along the space, it becomes easier for the joint generation algorithm to find a new vector not in the already generated sets $\mathcal{F}^* \subseteq \mathcal{F}_\pi$ and $\mathcal{G}^* \subseteq \mathcal{G}_\pi$.

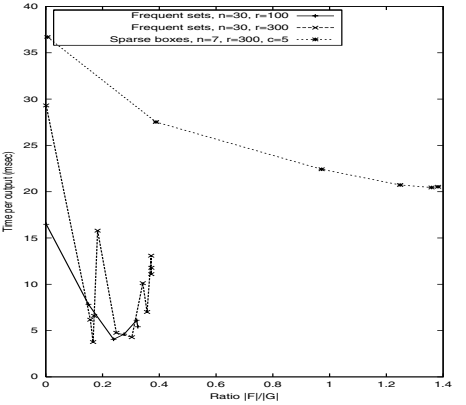

Fig. 5. Average generation time as a function of the ratio $|\mathcal{G}_\pi|/|\mathcal{F}_\pi|$, for properties π_2 and π_3.

6 Conclusion

We have presented an efficient implementation for a quasi-polynomial algorithm for jointly generating the families $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ of minimal satisfying and maximal non-satisfying vectors for a given monotone property π. We provided experimental evaluation of the algorithm on three different monotone properties. Our experiments indicate that the algorithm behaves much more efficiently than its worst-case time complexity indicates. The algorithm seems to run faster on instances where the families $\mathcal{F}_\pi$ and $\mathcal{G}_\pi$ are not very biased in size. Finally, our experiments also indicate that such non-bias in size is not a rare situation (for random instances), despite the fact that inequalities of the form (3)-(5) may hold in general.

References

1. M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge Univ. Press, 1992.
2. R. Agrawal, T. Imielinski and A. Swami, Mining associations between sets of items in massive databases, *Proc. 1993 ACM-SIGMOD Int. Conf.*, pp. 207-216.
3. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, Fast discovery of association rules, in *Advances in Knowledge Discovery and Data Mining* (U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, eds.), pp. 307-328, AAAI Press, Menlo Park, California, 1996.
4. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K.Makino, Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities, *SIAM Journal on Computing*, **31** (5) (2002) pp. 1624-1643.
5. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, Generating Dual-Bounded Hypergraphs, *Optimization Methods and Software*, (OMS) **17** (5), Part I (2002), pp. 749–781.
6. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K. Makino, An Intersection Inequality for Discrete Distributions and Related Generation Problems, in *Automata, Languages and Programming, 30-th International Colloquium, ICALP 2003*, *Lecture Notes in Computer Science (LNCS)* 2719 (2003) pp. 543–555.
7. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, An Efficient Implementation of a Quasi-Polynomial Algorithm for Generating Hypergraph Transversals, in *the Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, *LNCS* 2832, pp. 556–567, Budapest, Hungary, September, 2003.
8. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, On enumerating minimal dicuts and strongly connected subgraphs, to appear in *the 10th Conference on Integer Programming and Combinatorial Optimization (IPCO X)*, DIMACS Technical Report 2003-35, Rutgers University, http://dimacs.rutgers.edu/TechnicalReports/2003.html.
9. E. Boros, V. Gurvich, L. Khachiyan and K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets, in *19th Int. Symp. on Theoretical Aspects of Computer Science, (STACS)*, March 2002, LNCS 2285, pp. 133–141.
10. J. C. Bioch and T. Ibaraki (1995). Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, **123**, pp. 50–63.
11. B. Chazelle, R. L. (Scot) Drysdale III and D. T. Lee, Computing the largest empty rectangle, *SIAM Journal on Computing*, 15(1) (1986) 550-555.

12. C. J. Colbourn, *The combinatorics of network reliability*, Oxford Univ. Press, 1987.
13. J. Edmonds, J. Gryz, D. Liang and R. J. Miller, Mining for empty rectangles in large data sets, in *Proc. 8th Int. Conf. on Database Theory (ICDT)*, Jan. 2001, *Lecture Notes in Computer Science* 1973, pp. 174–188.
14. T. Eiter and G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24 (1995) pp. 1278-1304.
15. M. L. Fredman and L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, *Journal of Algorithms*, 21 (1996) pp. 618–628.
16. V. Gurvich and L. Khachiyan, On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions, *Discrete Applied Mathematics*, 96-97 (1999) pp. 363-373.
17. D. Gunopulos, R. Khardon, H. Mannila and H. Toivonen, Data mining, hypergraph transversals and machine learning, in *Proc. 16th ACM-PODS Conf.*, (1997) pp. 209–216.
18. V. Gurvich, To theory of multistep games, *USSR Comput. Math. and Math Phys.* **13-6** (1973), pp. 1485–1500.
19. V. Gurvich, Nash-solvability of games in pure strategies, *USSR Comput. Math. and Math. Phys.*, **15** (1975), pp. 357–371.
20. E. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, *SIAM Journal on Computing*, 9 (1980), pp. 558–565.
21. B. Liu, L.-P. Ku and W. Hsu, Discovering interesting holes in data, in *Proc. IJCAI*, pp. 930–935, Nagoya, Japan, 1997.
22. B. Liu, K. Wang, L.-F. Mun and X.-Z. Qi, Using decision tree induction for discovering holes in data, in *Proc. 5th Pacific Rim Int. Conf. on Artificial Intelligence*, pp. 182-193, 1998.
23. M. Orlowski, A new algorithm for the large empty rectangle problem, *Algorithmica* 5(1) (1990) 65-73.