Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 154 (2006) 2350–2372
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation☆

Leonid Khachiyan[a,✠], Endre Boros[b], Khaled Elbassioni[c], Vladimir Gurvich[b]

[a] *Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8004, USA*
[b] *RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA*
[c] *Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany*

Received 9 June 2004; accepted 21 September 2005
Available online 6 June 2006

Abstract

Given a finite set V, and a hypergraph $\mathscr{H} \subseteq 2^V$, the hypergraph transversal problem calls for enumerating all minimal hitting sets (transversals) for $\mathscr{H}$. This problem plays an important role in practical applications as many other problems were shown to be polynomially equivalent to it. Fredman and Khachiyan [On the complexity of dualization of monotone disjunctive normal forms, J. Algorithms 21 (1996) 618–628] gave an incremental quasi-polynomial-time algorithm for solving the hypergraph transversal problem. In this paper, we present an efficient implementation of this algorithm. While we show that our implementation achieves the same theoretical worst-case bound, practical experience with this implementation shows that it can be substantially faster. We also show that a slight modification of the original algorithm can be used to obtain a stronger bound on the running time.

More generally, we consider a monotone property π over a bounded n-dimensional integral box. As an important application of the above hypergraph transversal problem, pioneered by Bioch and Ibaraki [Complexity of identification and dualization of positive Boolean functions, Inform. and Comput. 123 (1995) 50–63], we consider the problem of incrementally generating simultaneously all minimal subsets satisfying π and all maximal subsets not satisfying π, for properties 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 via a polynomial-time reduction to a generalization of the hypergraph transversal problem on integer boxes. In this paper we present an efficient implementation of this procedure, and present experimental results to evaluate our implementation for a number of interesting monotone properties π.

Keywords: Dualization; Hypergraph transversals; Generation algorithms; Monotone properties; Frequent sets; Empty boxes

✠Our friend and colleague, Leo Khachiyan passed away with tragic suddenness while this paper was under review.

E-mail addresses: boros@rutcor.rutgers.edu (E. Boros), elbassio@mpi-sb.mpg.de (K. Elbassioni), gurvich@rutcor.rutgers.edu (V. Gurvich).

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

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.04.012

1. Introduction

Let V be a finite set of cardinality $|V| = n$. For a hypergraph $\mathscr{H} \subseteq 2^V$, let us denote by $\mathscr{I}(\mathscr{H})$ the family of its *maximal independent* sets, i.e. maximal subsets of V not containing any hyperedge of $\mathscr{H}$. The complement of a maximal independent subset is called a *minimal transversal* of $\mathscr{H}$ (i.e. minimal subset of V intersecting all hyperedges of $\mathscr{H}$). The collection $\mathscr{H}^{\mathrm{d}}$ of minimal transversals is also called the *dual* or *transversal* hypergraph for $\mathscr{H}$. The *hypergraph transversal problem* is the problem of generating all transversals of a given hypergraph. This problem has important applications in combinatorics [26], artificial intelligence [15], game theory [18,19], reliability theory [13], database theory [11,15,17], integer programming [8], learning theory [3], and data mining [1,9,11].

The *theoretically* best known algorithm for solving the hypergraph transversal problem is due to Fredman and Khachiyan [16] and works by performing $|\mathscr{H}^{\mathrm{d}}| + 1$ calls to the following problem, known as *hypergraph dualization*:

DUAL ($\mathscr{H}, \mathscr{X}$): *Given a complete list of all hyperedges of* $\mathscr{H}$, *and a set of minimal transversals* $\mathscr{X} \subseteq \mathscr{H}^{\mathrm{d}}$, *either prove that* $\mathscr{X} = \mathscr{H}^{\mathrm{d}}$, *or find a new transversal* $X \in \mathscr{H}^{\mathrm{d}} \backslash \mathscr{X}$.

Two recursive algorithms were proposed in [16] to solve the hypergraph dualization problem. These algorithms have incremental quasi-polynomial-time complexities of $poly(n) + m^{\mathrm{O}(\log^2 m)}$ and $poly(n) + m^{\mathrm{o}(\log m)}$, respectively, where $m = |\mathscr{H}| + |\mathscr{X}|$. Even though the second algorithm is theoretically more efficient, the first algorithm is much simpler in terms of its implementation overhead, making it more attractive for practical applications. In fact, as we have found out experimentally, in many cases, the most critical parts of the dualization procedure, in terms of execution time, are operations performed in each recursive call, rather than the total number of recursive calls. With respect to this measure, the first algorithm is more efficient due to its simplicity. For that reason, we present in this paper an implementation of the first algorithm in [16], which is efficient with respect to the time per recursive call. We further show that this efficiency in implementation does not come at the cost of increasing the worst-case running time substantially.

Rather than considering the hypergraph dualization problem, we shall consider, in fact, the more general problem of dualization on boxes introduced in [8]. In this latter problem, we are given an integral box $\mathscr{C} = \mathscr{C}_1 \times \cdots \times \mathscr{C}_n$, where $\mathscr{C}_i = \{\ell_i, \ell_i + 1, \ldots, u_i\}$ is a finite set of consecutive integers, and we let $\ell = (\ell_1, \ldots, \ell_n)$ and $u = (u_1, \ldots, u_n)$ denote the unique minimal and maximal elements in $\mathscr{C}$ with respect to the usual componentwise comparison. For a subset $\mathscr{A} \subseteq \mathscr{C}$ we denote by $\mathscr{A}^+ = \{x \in \mathscr{C} | x \geqslant a, \text{ for some } a \in \mathscr{A}\}$ and $\mathscr{A}^- = \{x \in \mathscr{C} | x \leqslant a, \text{ for some } a \in \mathscr{A}\}$, the ideal and filter generated by $\mathscr{A}$. For single element subsets $\mathscr{A} = \{a\}$ we shall write a^+ and a^- instead of $\{a\}^+$ and $\{a\}^-$, respectively. Any element in $\mathscr{C} \backslash \mathscr{A}^+$ is called *independent of* $\mathscr{A}$, and we let $\mathscr{I}(\mathscr{A})$ denote the set of all maximal independent elements for $\mathscr{A}$. Call a family of vectors $\mathscr{A}$ *Sperner* if $\mathscr{A}$ is an antichain, i.e. if no two elements are comparable in $\mathscr{A}$. If $\mathscr{C}$ is the Boolean cube $2^{[n]}$, we get the well-known definitions of a hypergraph $\mathscr{A}$ and its family of maximal independent sets $\mathscr{I}(\mathscr{A})$. Given $\mathscr{A} \subseteq \mathscr{C}$ and a subset $\mathscr{B} \subseteq \mathscr{I}(\mathscr{A})$ of maximal independent elements of $\mathscr{A}$, problem DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$) calls for generating a new element $x \in \mathscr{I}(\mathscr{A}) \backslash \mathscr{B}$, or proving that there is no such element:

DUAL ($\mathscr{C}, \mathscr{A}, \mathscr{B}$): *Given a family of vectors* $\mathscr{A} \subseteq \mathscr{C}$, *and a subset* $\mathscr{B} \subseteq \mathscr{I}(\mathscr{A})$ *of its maximal independent vectors, either find a new maximal independent vector* $x \in \mathscr{I}(\mathscr{A}) \backslash \mathscr{B}$, *or prove that* $\mathscr{B} = \mathscr{I}(\mathscr{A})$.

By performing $|\mathscr{I}(\mathscr{A})| + 1$ calls to problem DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$), we can solve the following problem.

GEN ($\mathscr{C}, \mathscr{I}(\mathscr{A})$): *Given an integral box* $\mathscr{C}$, *and a subset of vectors* $\mathscr{A} \subseteq \mathscr{C}$, *generate all maximal independent elements of* $\mathscr{A}$.

Problem GEN($\mathscr{C}, \mathscr{I}(\mathscr{A})$) has several interesting applications in integer programming and data mining, see [8,6,9] and the references therein. In particular, it is known that the incremental generation of minimal infrequent elements of a given database of quantitative attributes, and the incremental generation of maximal boxes not containing any point from a given point set, both reduce in polynomial time to problem GEN($\mathscr{C}, \mathscr{I}(\mathscr{A})$), see [6,9]. These problems are motivated, respectively, by the generation of association rules in databases with quantitative attributes [27], and by the discovery of missing associations or "holes" in data mining applications (see [23,24]).

Extensions of the above-mentioned hypergraph transversal algorithms to solve problem DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$) were given in [8]. In this paper, we give an implementation of the first dualization algorithm in [8], which achieves efficiency in two directions:

- Reuse of the recursion tree: dualization-based techniques generate all maximal independent elements of a given subset $\mathscr{A} \subseteq \mathscr{C}$ by usually performing $|\mathscr{I}(\mathscr{A})| + 1$ calls to problem DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$), thus building a new recursion tree for each call. However, as it will be illustrated, it is more efficient to use the same recursion tree to

generate all the elements of $\mathscr{I}(\mathscr{A})$, since the recursion trees required to generate many elements may be nearly identical.

- Efficient implementation at each recursion tree node: straightforward implementation of the algorithm in [8] requires $O(n|\mathscr{A}| + n|\mathscr{B}|)$ time per recursive call. However, this can be improved to $O(n|\mathscr{A}| + |\mathscr{B}| + n\log^3(n|\mathscr{B}|))$ by maintaining a binary search tree on the elements of $\mathscr{B}$, and using randomization. Since in many natural examples $|\mathscr{B}|$ is much larger than $|\mathscr{A}|$, this gives a significant improvement. Moreover, the expected memory required by the randomized version is only $O(nm)$ whereas the deterministic version may require up to $O(nm^2)$ space, where $m = |\mathscr{A}| + |\mathscr{B}|$.

Several heuristics are also used to improve the running time. For instance, we use random sampling to find the branching variable and its value, required to divide the problem at the current recursion node. We also estimate the numbers of elements of $\mathscr{A}$ and $\mathscr{B}$ that are active at the current node, and only actually compute these active elements when their numbers drop by a certain factor. As our experiments indicate, such heuristics can effectively reduce the running time of the algorithm.

Let $\pi : \mathscr{C} \mapsto \{0, 1\}$ be a *monotone* property defined over the elements of an integral box $\mathscr{C} = \mathscr{C}_1 \times \cdots \times \mathscr{C}_n$: if $x \in \mathscr{C}$ satisfies property π, i.e. $\pi(x) = 1$, then any $y \in \mathscr{C}$ such that $y \geqslant x$ also satisfies π. We assume that π is described by a polynomial satisfiability oracle $\mathscr{O}_\pi$, i.e. an algorithm that can decide whether a given vector $x \in \mathscr{C}$ satisfies π, in time polynomial in n and the size $|\pi|$ of the input description of π. Denote, respectively, by $\mathscr{F}_\pi \subseteq \mathscr{C}$ and $\mathscr{G}_\pi \subseteq \mathscr{C}$ the families of minimal elements satisfying property π, and maximal elements not satisfying property π. Then it is clear that $\mathscr{G}_\pi = \mathscr{I}(\mathscr{F}_\pi)$ for any monotone property π. Given a monotone property π, we consider the problem of jointly generating the families $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$:

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

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

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

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

Problems GEN($\mathscr{C}, \mathscr{F}_\pi, \mathscr{X}$) and GEN($\mathscr{C}, \mathscr{G}_\pi, \mathscr{Y}$) arise in many practical applications and in a variety of fields, see e.g. [8,9,7,11,15,22]. It is easy to see that problem GEN($\mathscr{C}, \mathscr{G}_\pi, \mathscr{X}$) includes as a special case the dualization problem DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$). Indeed, for any $x \in \mathscr{C}$, if we define the monotone property $\pi(x)$ to be satisfied by x if and only if $x \geqslant a$ for some $a \in \mathscr{A}$, then the families $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$ can be identified with the families $\mathscr{A}$ and $\mathscr{I}(\mathscr{A})$, respectively.

Even though the above two generation problems may be NP-hard in general (see e.g. [22]), it is known that the joint generation problem GEN($\mathscr{C}, \mathscr{F}_\pi, \mathscr{G}_\pi, \mathscr{X}, \mathscr{Y}$) can always be solved in incremental $poly(n, \log \|\mathscr{C}\|_\infty) + m^{\text{polylog } m}$ time (i.e. in incremental quasi-polynomial time) for any monotone property π described by a satisfiability oracle, where $\|\mathscr{C}\|_\infty = \sum_{i=1}^n |\mathscr{C}_i|$ and $m = |\mathscr{X}| + |\mathscr{Y}|$, see [5,8,20]. In particular, as observed by Bioch and Ibaraki [5], and independently by Gurvich and Khachiyan [20], there is a polynomial-time reduction from the joint generation problem to the dualization problem on boxes (see also [8]):

Proposition 1. *Problem* GEN($\mathscr{C}, \mathscr{F}_\pi, \mathscr{G}_\pi, \mathscr{A}, \mathscr{B}$) *can be solved in time* $\sum_{i=1}^n poly(\log \|\mathscr{C}\|_\infty, |\mathscr{A}|, |\mathscr{B}|) + T(\|\mathscr{O}_\pi\|) + T_{\text{dual}}$ *for any monotone property π defined by a satisfiability oracle $\mathscr{O}_\pi$, where $T(\|\mathscr{O}_\pi\|)$ is the worst-case running time of the oracle on any $x \in \mathscr{C}$, and T_{dual} denotes the time required to solve problem* DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$).

Proof. Let us first consider two subroutines which can be defined for any monotone property π. The first of these subroutines takes as input a vector $x \in \mathscr{F}_\pi^+$ and returns a minimal vector x^* in $\mathscr{F}_\pi^+ \cap x^-$. Such a vector $x^* = \min_{\mathscr{F}_\pi}(x)$

can, for instance, be computed by coordinate descent:

$$
\begin{aligned}
x_1^* &\leftarrow \min\{y_1 \in \mathscr{C}_1 \mid (y_1, x_2, \ldots, x_{n-1}, x_n) \in \mathscr{F}_\pi^+,\ y_1 \leqslant x_1\},\\
x_2^* &\leftarrow \min\{y_2 \in \mathscr{C}_2 \mid (x_1^*, y_2, \ldots, x_{n-1}, x_n) \in \mathscr{F}_\pi^+,\ y_2 \leqslant x_2\},\\
&\cdots\\
x_n^* &\leftarrow \min\{y_n \in \mathscr{C}_n \mid (x_1^*, x_2^*, \ldots, x_{n-1}^*, y_n) \in \mathscr{F}_\pi^+,\ y_n \leqslant x_n\},
\end{aligned}
$$

where $\min\{y_i \in \mathscr{C}_i \mid \cdots\}$ returns an arbitrary minimal element with the specified property. Note that each of the n coordinate steps in the above procedures can be reduced via binary search to at most $\log(\|\mathscr{C}\|_\infty + 1)$ satisfiability oracle calls for the monotone property π.

The second subroutine is to compute, for a given vector $x \in \mathscr{I}(\mathscr{F}_\pi)^-$, a maximal vector $x^* = \max_{\mathscr{F}_\pi}(x) \in \mathscr{I}(\mathscr{F}_\pi)^- \cap x^+$ which can be computed analogously by coordinate ascent. It is clear that these two subroutines can be executed in at most $O(\sum_{i=1}^n \log(\|\mathscr{C}\|_\infty + 1) \cdot T(\|\mathscr{O}_\pi\|))$ time.

Now the result stated in the proposition is obtained via the following algorithm.

Algorithm $\mathscr{J}$

Step 1: Check whether $\mathscr{B} \subseteq \mathscr{I}(\mathscr{A})$. Since $\mathscr{B} \subseteq \mathscr{I}(\mathscr{F}_\pi)$ and $\mathscr{A} \subseteq \mathscr{F}_\pi$, each vector $x \in \mathscr{B}$ is independent of $\mathscr{A}$ and we only need to check the maximality of x for $\mathscr{I}(\mathscr{A})$. In other words, we have to check whether or not $y \in \mathscr{A}^+$ for every immediate successor y of x. Since both $\mathscr{A}$ and $\mathscr{B}$ are explicitly given, this check can be done in $poly(n, |\mathscr{A}|, |\mathscr{B}|)$ comparisons. If there is an $x \in \mathscr{B}\backslash\mathscr{I}(\mathscr{A})$, then $x \notin \mathscr{F}_\pi^+$ because $x \in \mathscr{B} \subseteq \mathscr{I}(\mathscr{F}_\pi)$. This and the inclusion $\mathscr{A} \subseteq \mathscr{F}_\pi$ imply that $x \notin \mathscr{A}^+$. Since $x \notin \mathscr{I}(\mathscr{A})$, we can find an immediate successor $y \notin \mathscr{A}^+$ of x. By the maximality of x in $\mathscr{C}\backslash\mathscr{F}_\pi^+$, y belongs to $\mathscr{F}_\pi^+$. Now letting $y^* = \min_{\mathscr{F}_\pi}(y)$ we conclude that $y^* \in \mathscr{F}_\pi\backslash\mathscr{A}$, i.e. y^* is a new minimal vector in $\mathscr{F}_\pi$.

Step 2: (is similar to the previous step): Let $\mathscr{I}^{-1}(\mathscr{B})$ be the set of minimal elements in $\mathscr{C}\backslash\mathscr{B}^-$. We check whether $\mathscr{A} \subseteq \mathscr{I}^{-1}(\mathscr{B})$. If $\mathscr{A}$ contains an element that is not minimal in $\mathscr{C}\backslash\mathscr{B}^-$, we can find a new vector in $\mathscr{I}(\mathscr{F}_\pi)\backslash\mathscr{B}$ and halt.

Step 3: Suppose that $\mathscr{B} \subseteq \mathscr{I}(\mathscr{A})$ and $\mathscr{A} \subseteq \mathscr{I}^{-1}(\mathscr{B})$. Then

$$(\mathscr{A}, \mathscr{B}) = (\mathscr{F}_\pi, \mathscr{I}(\mathscr{F}_\pi)) \iff \mathscr{B} = \mathscr{I}(\mathscr{A}).$$

To see this, assume that $\mathscr{B} = \mathscr{I}(\mathscr{A})$, and suppose on the contrary that there is an $x \in \mathscr{F}_\pi\backslash\mathscr{A}$. Since $x \notin \mathscr{A} = \mathscr{I}^{-1}(\mathscr{B})$ and $x \notin \mathscr{B}^- \subseteq \mathscr{I}(\mathscr{F}_\pi)^-$, there must exist a $y \in \mathscr{A} \subseteq \mathscr{F}_\pi$ such that $y \leqslant x$. Hence we get two distinct elements $x, y \in \mathscr{F}_\pi$ such that $y \leqslant x$, which contradicts the definition of $\mathscr{F}_\pi$. The existence of an $x \in \mathscr{I}(\mathscr{F}_\pi)\backslash\mathscr{B}$ leads to a similar contradiction.

To check the stopping criterion $\mathscr{B} = \mathscr{I}(\mathscr{A})$, we solve problem DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$). If $\mathscr{B} \neq \mathscr{I}(\mathscr{A})$, we obtain a new point $x \in \mathscr{I}(\mathscr{A})\backslash\mathscr{B}$. By $\mathscr{F}_\pi^+ \cap \mathscr{I}(\mathscr{F}_\pi)^- = \emptyset$, either $x \in \mathscr{F}_\pi^+$, or $x \in \mathscr{I}(\mathscr{F}_\pi)^-$ and we can decide which of these two cases holds by calling the satisfiability oracle $\mathscr{O}_\pi$ on x. In the first case, we conclude that $x^* = \min_{\mathscr{F}_\pi}(x)$ is a new vector in $\mathscr{F}_\pi\backslash\mathscr{A}$, while in the second case we have $x^* = \max_{\mathscr{F}_\pi}(x) \in \mathscr{I}(\mathscr{F}_\pi)\backslash\mathscr{B}$ as a new vector. □

As mentioned above, the currently best known algorithm for dualization runs in time $poly(n) + m^{o(\log m)}$, see [8,16]. Unfortunately, this joint generation may not be an efficient algorithm for solving either of GEN($\mathscr{C}, \mathscr{F}_\pi, \mathscr{X}$) or GEN($\mathscr{C}, \mathscr{G}_\pi, \mathscr{Y}$) separately for the simple reason that we do not control which of the families $\mathscr{F}_\pi\backslash\mathscr{X}$ and $\mathscr{G}_\pi\backslash\mathscr{Y}$ contains each new vector produced by the algorithm. Suppose we want to generate $\mathscr{F}_\pi$ and the family $\mathscr{G}_\pi$ is exponentially larger than $\mathscr{F}_\pi$. Then, if we are unlucky, we may get elements of $\mathscr{F}_\pi$ with exponential delay, while getting large subfamilies of $\mathscr{G}_\pi$ (which are not needed at all) in between. However, there are two reasons why 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 $\mathscr{F}_\pi$ in fewer than $|\mathscr{F}_\pi| + |\mathscr{G}_\pi|$ steps, in general:

Proposition 2. *Consider an arbitrary algorithm A, which generates the family $\mathscr{F}_\pi$ by using only a satisfiability oracle $\mathscr{O}_\pi$ for the monotone property π. Then, for the algorithm to generate the whole family $\mathscr{F}_\pi$, it must call the oracle at least $|\mathscr{F}_\pi| + |\mathscr{G}_\pi|$ times.*

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

$$\pi_y(x) = \begin{cases} \pi(x) & \text{if } x \in \mathscr{C}, x \neq y \\ 1 - \pi(x) & \text{if } x = y. \end{cases}$$

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

Second, for a wide class of Sperner families (or equivalently, monotone properties), the so-called *uniformly dual-bounded* families [10], it was realized that the size of the dual family $\mathscr{I}(\mathscr{F}_\pi)$ is uniformly bounded by a (quasi-) polynomial in the size of $\mathscr{F}_\pi$ and the oracle description:

$$|\mathscr{I}(\mathscr{X}) \cap \mathscr{I}(\mathscr{F}_\pi)| \leqslant (\textit{quasi-})\textit{poly}(|\mathscr{X}|, |\pi|) \tag{1}$$

for any non-empty subfamily $\mathscr{X} \subseteq \mathscr{F}_\pi$ (see e.g. [8,6]). An inequality of the form (1) implies that joint generation is an incrementally efficient way for generating the family $\mathscr{F}_\pi$ (see Section 3.1 and also [6] for several interesting examples):

Corollary 1. *Suppose $\mathscr{F}_\pi$ is uniformly dual-bounded as in* (1) *and defined by a* (*quasi-*) *polynomial-time membership oracle for $\mathscr{F}_\pi^+$. Then problem* GEN($\mathscr{C}, \mathscr{F}_\pi, \mathscr{X}$) *is* (*quasi-*) *polynomial-time reducible to at most* (*quasi-*) *poly*($|\mathscr{X}|, \log(\|\mathscr{C}\|_\infty + 1), \|\mathscr{O}_\pi\|) + 1$ *instances of problem* DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$).

Proof. Given a subset $\mathscr{X}$ of $\mathscr{F}_\pi$, we repeatedly run Algorithm $\mathscr{J}$, starting with $\mathscr{A} = \mathscr{X}$ and $\mathscr{B} = \emptyset$, until it either produces a new element in $\mathscr{F}_\pi \backslash \mathscr{X}$ or proves that $\mathscr{X} = \mathscr{F}_\pi$ by generating the entire family $\mathscr{I}(\mathscr{F}_\pi)$. By Step 1, either $\mathscr{B} \subseteq \mathscr{I}(\mathscr{X})$ is maintained during the execution of the algorithm, or a new element $x \in \mathscr{F}_\pi \backslash \mathscr{X}$ can be found. Thus, as long as Algorithm $\mathscr{J}$ outputs elements of $\mathscr{I}(\mathscr{F}_\pi)$, these elements also belong to $\mathscr{I}(\mathscr{X})$, and hence the total number of such elements does not exceed (*quasi-*)*poly*($|\mathscr{X}|, |\pi|$). □

In Section 2.2, we present an efficient implementation of a quasi-polynomial dualization algorithm on boxes. Direct application of this implementation to solve the joint generation problem, as suggested by Proposition 1, 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 $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$. The details of this method will be described in Section 3.2. In Section 2.1 we introduce some basic terminology used throughout the paper, and briefly outline the quasi-polynomial algorithm of [16] (or more precisely, its generalization to boxes [8]). Section 2.2 describes the data structure used in our implementation of this algorithm, and presents the algorithm. In Section 2.3, we show that the new version of the algorithm has, on the average, the same quasi-polynomial bound on the running time as that of [8], and we also show how to get a slightly stronger bound on the running time. In Section 3.1, we give three examples of monotone properties that will be used in our experimental study of the joint generation algorithm. Finally, Section 4 presents our experimental findings, and Section 5 provides some conclusions.

2. Dualization on integer boxes

2.1. Terminology and outline of the algorithm

Throughout this section, we assume that we are given an integer box $\mathscr{C}^* = \mathscr{C}_1^* \times \cdots \times \mathscr{C}_n^*$, where $\mathscr{C}_i^* = [l_i^* : u_i^*]$, and $l_i^* \leqslant u_i^*$, are integers, and a subset $\mathscr{A}^* \subseteq \mathscr{C}^*$ of vectors for which it is required to generate all maximal independent elements. The algorithm of [8], considered in this paper, solves problem DUAL($\mathscr{C}^*, \mathscr{A}^*, \mathscr{B}^*$), 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 $\mathscr{C} = \mathscr{C}_1 \times \cdots \times \mathscr{C}_n$ of the original box $\mathscr{C}^*$ and two subsets $\mathscr{A} \subseteq \mathscr{A}^*$ and $\mathscr{B} \subseteq \mathscr{B}^*$ of integral vectors, where $\mathscr{B}^* \subseteq \mathscr{I}(\mathscr{A}^*)$ denotes the subfamily of maximal independent elements that the algorithm has generated so far. Note that, by definition, the following condition holds for the original problem and all subsequent subproblems:

$$a \not\leqslant b \quad \text{for all } a \in \mathscr{A},\ b \in \mathscr{B}. \tag{2}$$

Given an element $a \in \mathscr{A}$ ($b \in \mathscr{B}$), we say that a coordinate $i \in [n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$ is *essential* for a (respectively, b), in the box $\mathscr{C} = [l_1 : u_1] \times \cdots \times [l_n : u_n]$, 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 $x \in \mathscr{A} \cup \mathscr{B}$. Finally, given a sub-box $\mathscr{C} \subseteq \mathscr{C}^*$, and two subsets $\mathscr{A} \subseteq \mathscr{A}^*$ and $\mathscr{B} \subseteq \mathscr{B}^*$, we shall say that $\mathscr{B}$ is *dual to* $\mathscr{A}$ in $\mathscr{C}$ if $\mathscr{A}^+ \cup \mathscr{B}^- \supseteq \mathscr{C}$.

A key lemma, on which the algorithm in [8] is based, is that either (i) there is an element $x \in \mathscr{A} \cup \mathscr{B}$ with at most $1/\varepsilon$ essential coordinates, where $\varepsilon \stackrel{\text{def}}{=} 1/(1+\log m)$ and $m \stackrel{\text{def}}{=} |\mathscr{A}|+|\mathscr{B}|$, or (ii) one can easily find a new maximal independent element $z \in \mathscr{C}$, by picking each element z_i independently at random from $\{l_i, u_i\}$ for $i=1, \ldots, n$; see subroutine Random solution$(\cdot, \cdot, \cdot)$ 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 $x \in \mathscr{A}$ has at most $1/\varepsilon$ essential coordinates. Then, by (2), there is an $i \in [n]$ such that $|\{b \in \mathscr{B} : b_i < x_i\}| \geqslant \varepsilon|\mathscr{B}|$. This allows us to decompose the original problem into two subproblems DUAL$(\mathscr{C}', \mathscr{A}, \mathscr{B}')$ and DUAL$(\mathscr{C}'', \mathscr{A}'', \mathscr{B})$, where $\mathscr{C}' = \mathscr{C}_1 \times \cdots \times \mathscr{C}_{i-1} \times [x_i : u_i] \times \mathscr{C}_{i+1} \times \cdots \times \mathscr{C}_n$, $\mathscr{B}' = \mathscr{B} \cap (\mathscr{C}')^+$, $\mathscr{C}'' = \mathscr{C}_1 \times \cdots \times \mathscr{C}_{i-1} \times [l_i : x_i - 1] \times \mathscr{C}_{i+1} \times \cdots \times \mathscr{C}_n$, and $\mathscr{A}'' = \mathscr{A} \cap (\mathscr{C}'')^-$ (note that although $\mathscr{A}^*, \mathscr{B}^* \subseteq \mathscr{C}^*$ holds initially, this property may not hold after one ore more decomposition steps). This way, the algorithm is guaranteed to reduce the cardinality of one of the sets $\mathscr{A}$ or $\mathscr{B}$ by a factor of at least $1 - \varepsilon$ 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 $\mathscr{B}'$, $\mathscr{A}''$ (see subroutine Est$(\cdot, \cdot)$ below). Second, once we have determined the new sub-boxes $\mathscr{C}'$, $\mathscr{C}''$ above, we do not compute the *active* families $\mathscr{B}'$ and $\mathscr{A}''$ 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 $\frac{1}{2}$, 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 $\mathrm{O}(nm + m/(1 - f))$, which is linear in m for constant f.

2.2. The dualization algorithm

We used the following data structures in our implementation:

- Two global arrays of vectors, A and B containing the elements of $\mathscr{A}^*$ and $\mathscr{B}^*$, respectively. We keep with every vector a counter indicating the number of essential components of that vector with respect to the current box. These counters are updated in $\mathrm{O}(|\mathscr{A}| + |\mathscr{B}|)$, after each decomposition step, where $\mathscr{A}$ and $\mathscr{B}$ are the current active sets.
- Two local (dynamic) arrays of indices, index$(\mathscr{A})$ and index$(\mathscr{B})$, containing the indices of vectors from $\mathscr{A}^*$ and $\mathscr{B}^*$ (i.e. containing pointers to elements of the arrays A and B), that appear in the current subproblem. These arrays are used to enable sampling from the sets $\mathscr{A}$ and $\mathscr{B}$, and also to keep track of which vectors are currently active, i.e. intersect the current box.
- A global balanced binary search tree $\mathbf{H}(\mathscr{B}^*)$, built on the elements of $\mathscr{B}^*$ using lexicographic ordering. Each node of the tree contains an index of an element in the array B. This way, checking whether a given vector $x \in \mathscr{C}$ belongs to $\mathscr{B}^*$ or not, takes only $\mathrm{O}(n \log |\mathscr{B}^*|)$ time.

In the sequel, we let $m = |\mathscr{A}| + |\mathscr{B}|$ and $\varepsilon = 1/(1 + \log m)$. We assume further that operations of the form $\mathscr{A}'' \leftarrow \mathscr{A}$ and $\mathscr{B}' \leftarrow \mathscr{B}$ are actually performed on the index arrays index(A), index(B) so that they only take $\mathrm{O}(m)$ rather than $\mathrm{O}(nm)$ time. We used the following subroutines in our implementation:

Maximization $\max_{\mathscr{A}}(z)$: It takes as input a vector $z \notin \mathscr{A}^+$ and returns a maximal vector z^* in $(\mathscr{C}^* \cap \{z\}^+) \backslash \mathscr{A}^+$. This can be done in $\mathrm{O}(n|\mathscr{A}|)$ by initializing $c(a) = |\{i \in [n] : a_i > z_i\}|$ for all $a \in \mathscr{A}$, and repeating, for $i = 1, \ldots, n$, the following two steps: (i) $z_i^* \leftarrow \min(u_i^*, \min\{a_i - 1 : a \in \mathscr{A}, c(a) = 1 \text{ and } a_i > z_i\})$ (where we assume $\min(\emptyset) = \infty$); (ii) $c(a) \leftarrow c(a) - 1$ for each $a \in \mathscr{A}$ such that $z_i < a_i \leqslant z_i^*$.

Exhaustive duality $(\mathscr{C}, \mathscr{A}, \mathscr{B})$: Assuming $|\mathscr{A}||\mathscr{B}| \leqslant 1$, check duality in $\mathrm{O}(n(|\mathscr{A}^*| + \log |\mathscr{B}|))$ as follows: first, if $|\mathscr{A}| = |\mathscr{B}| = 1$ then find an $i \in [n]$ such that $a_i > b_i$, where $\mathscr{A} = \{a\}$ and $\mathscr{B} = \{b\}$. (Such a coordinate is guaranteed to exist by (2).) Let $l = (l_1, \ldots, l_n)$ and $u = (u_1, \ldots, u_n)$ be the minimum and maximum elements of $\mathscr{C}$, respectively. If there is a $j \neq i$ such that $b_j < u_j$ then return $\max_{\mathscr{A}^*}(u_1, \ldots, u_{i-1}, b_i, u_{i+1}, \ldots, u_n)$. If there is a $j \neq i$ such that $a_j > l_j$ then return $(u_1, \ldots, u_{j-1}, a_j - 1, u_{j+1}, \ldots, u_n)$. If $b_i < a_i - 1$ then return also $(u_1, \ldots, u_{i-1}, a_i - 1, u_{i+1}, \ldots, u_n)$. Otherwise return the empty vector Λ (meaning that $\mathscr{A}$ and $\mathscr{B}$ are dual in $\mathscr{C}$). Second, if $|\mathscr{A}| = 0$ then let $z = \max_{\mathscr{A}^*}(u)$, and return either Λ if $z \in \mathscr{B}^*$, or z if $z \notin \mathscr{B}^*$ (this check can be done in $\mathrm{O}(n \log |\mathscr{B}^*|)$ using the search tree $\mathbf{H}(\mathscr{B}^*)$). Finally, if $|\mathscr{B}| = 0$ then return either Λ or $z = \max_{\mathscr{A}^*}(l)$ depending on whether $l \in \mathscr{A}^+$ or not (this check requires $\mathrm{O}(n|\mathscr{A}|)$ time).

Random solution ($\mathscr{C}, \mathscr{A}^*, \mathscr{B}^*$): Repeat the following for $k=1,\ldots,t_1$, where t_1 is a constant (say 10): find a random point $z^k \in \mathscr{C}$, by picking each coordinate z_i^k randomly from $\{l_i, u_i\}, i=1,\ldots,n$. If $z^k \notin (\mathscr{A}^*)^+$, let $(z^k)^* \leftarrow \max_{\mathscr{A}^*}(z^k)$ and if $(z^k)^* \notin \mathscr{B}^*$ then return $(z^k)^*$. If no vector $(z^k)^*$ was returned in one of these t_1 iterations, then return Λ. This step takes $O(n(|\mathscr{A}^*| + \log|\mathscr{B}^*|))$ time, and is used to check whether $(\mathscr{A}^*)^+ \cup (\mathscr{B}^*)^-$ covers a large portion of $\mathscr{C}$.

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

Cleanup($\mathscr{A}$,$\mathscr{C}$) (Cleanup($\mathscr{B}$,$\mathscr{C}$)): Set $\mathscr{A}' \leftarrow \{a \in \mathscr{A} | a^+ \cap \mathscr{C} \neq \emptyset\}$ (respectively, $\mathscr{B}' \leftarrow \{b \in \mathscr{B} | b^- \cap \mathscr{C} \neq \emptyset\}$), and return $\mathscr{A}'$ (respectively, $\mathscr{B}'$). This step takes $O(n|\mathscr{A}|)$ (respectively, $O(n|\mathscr{B}|)$).

Now,we describe the implementation of procedure GEN-DUAL($\mathscr{A}, \mathscr{B}, \mathscr{C}$) which is called initially using $\mathscr{C} \leftarrow \mathscr{C}^*$,$\mathscr{A} \leftarrow \mathscr{A}^*$ and $\mathscr{B} \leftarrow \emptyset$. At the return of this call,$\mathscr{B}$ is extended by the elements in $\mathscr{I}(\mathscr{A}^*)$. Below we assume that $f \in (0,1)$ is a constant. We further assume that the procedure has access to the global data structure described earlier, and in particular to the original set of vectors $\mathscr{A}^*$ and $\mathscr{B}^*$.

Procedure GEN-DUAL($\mathscr{C}, \mathscr{A}, \mathscr{B}$):
Input: A box $\mathscr{C} = \mathscr{C}_1 \times \cdots \times \mathscr{C}_n$ and subsets $\mathscr{A} \subseteq \mathscr{A}^*$, and $\mathscr{B} \subseteq \mathscr{I}(\mathscr{A}^*)$.
Output: A subset $\mathscr{N} \subseteq \mathscr{I}(\mathscr{A}^*) \backslash \mathscr{B}$.

1. $\mathscr{N} \leftarrow \emptyset$.
2. While $|\mathscr{A}\|\mathscr{B}| \leqslant 1$
   2.1. $z \leftarrow$ Exhaustive duality($\mathscr{C}, \mathscr{A}, \mathscr{B}$).
   2.2. If $z = \Lambda$ then return($\mathscr{N}$).
   2.3. $\mathscr{B} \leftarrow \mathscr{B} \cup \{z\}$,$\mathscr{N} \leftarrow \mathscr{N} \cup \{z\}$.
   end while
3. $z \leftarrow$ Random solution($\mathscr{C}, \mathscr{A}^*, \mathscr{B}^*$).
4. While ($z \neq \Lambda$) do
   4.1.$\mathscr{B} \leftarrow \mathscr{B} \cup \{z\}$, $\mathscr{N} \leftarrow \mathscr{N} \cup \{z\}$.
   4.2. $z \leftarrow$ Random solution($\mathscr{C}, \mathscr{A}^*, \mathscr{B}^*$).
   end while
5. $x^* \leftarrow \mathrm{argmin}\{|\mathrm{Ess}(y)| : y \in (\mathscr{A} \cap \mathscr{C}^-) \cup (\mathscr{B} \cap \mathscr{C}^+)\}$.
6. If $x^* \in \mathscr{A}$ then
   6.1. $i \leftarrow \mathrm{argmax}\{\mathrm{Est}(\{b \in \mathscr{B} : b_j < x_j^*\}, \mathscr{C}) : j \in \mathrm{Ess}(x^*)\}$.
   6.2. $\mathscr{C}' = \mathscr{C}_1 \times \cdots \times \mathscr{C}_{i-1} \times [x_i^* : u_i] \times \mathscr{C}_{i+1} \times \cdots \times \mathscr{C}_n$.
   6.3. If $\mathrm{Est}(\mathscr{B}, \mathscr{C}') \leqslant f * |\mathscr{B}|$ then
      6.3.1. $\mathscr{B}' \leftarrow$ Cleanup($\mathscr{B}, \mathscr{C}'$).
   6.4. else
      6.4.1. $\mathscr{B}' \leftarrow \mathscr{B}$.
   6.5. $\mathscr{N}' \leftarrow$ GEN-DUAL($\mathscr{C}', \mathscr{A}, \mathscr{B}'$).
   6.6. $\mathscr{N} \leftarrow \mathscr{N} \cup \mathscr{N}'$,$\mathscr{B} \leftarrow \mathscr{B} \cup \mathscr{N}'$.
   6.7. $\mathscr{C}'' = \mathscr{C}_1 \times \cdots \times \mathscr{C}_{i-1} \times [l_i : x_i^* - 1] \times \mathscr{C}_{i+1} \times \cdots \times \mathscr{C}_n$.
   6.8. If $\mathrm{Est}(\mathscr{A}, \mathscr{C}'') \leqslant f * |\mathscr{A}|$ then
      6.8.1. $\mathscr{A}'' \leftarrow$ Cleanup($\mathscr{A}, \mathscr{C}''$).
   6.9. else
      6.9.1. $\mathscr{A}'' \leftarrow \mathscr{A}$.
   6.10. $\mathscr{N}'' \leftarrow$ GEN-DUAL($\mathscr{C}'', \mathscr{A}'', \mathscr{B}$).
   6.11. $\mathscr{N} \leftarrow \mathscr{N} \cup \mathscr{N}''$, $\mathscr{B} \leftarrow \mathscr{B} \cup \mathscr{N}''$.
7. else
   7.1–7.11. Symmetric versions of Steps 6.1–6.11 (details omitted).
   end if
8. Return ($\mathscr{N}$).

---

[1] Note that these sample sizes were chosen to theoretically get a guarantee on the expected running time of the algorithm. However, as our experiments indicate, smaller (usually constant) sample sizes are enough to provide practically good performance.

2.3. Analysis of the expected running time

Let $C(v)$ be the expected number of recursive calls on a subproblem GEN-DUAL($\mathscr{C}$, $\mathscr{A}$, $\mathscr{B}$) of volume $v \stackrel{\text{def}}{=} |\mathscr{A}\|\mathscr{B}|$. Consider a particular recursive call of the algorithm and let $\mathscr{A}$, $\mathscr{B}$ and $\mathscr{C}$ be the current inputs to this call. Let x^* be the element with minimum number of essential coordinates found in Step 5, and assume without loss of generality that $x^* \in \mathscr{A}$. As mentioned before, we assume also that the factor f used in Steps 6.3 and 6.8 is $\frac{1}{2}$. Since we do delayed cleanups, some of the vectors in $\mathscr{A}$ and $\mathscr{B}$ may not be active. For $i = 1, \ldots, n$, let $\mathscr{B}_i \stackrel{\text{def}}{=} \{b \in \mathscr{B} : b_i < x^*_i\}$, and denote by $\overline{\mathscr{B}} = \mathscr{B} \cap \mathscr{C}^+$ and $\overline{\mathscr{B}_i} = \mathscr{B}_i \cap \mathscr{C}^+$ the subsets of $\mathscr{B}$ and $\mathscr{B}_i$ that are active with respect to the current box $\mathscr{C}$. In this section, we show that our implementation has, with high probability, almost the same quasi-polynomial bound on the running time as the algorithm of [8].

Lemma 1. *Suppose that* $k \in \mathrm{Ess}(x^*)$ *satisfies* $|\overline{\mathscr{B}_k}| \geqslant \varepsilon|\overline{\mathscr{B}}|$. *Let* $i \in [n]$ *be the coordinate obtained in Step* 6.1 *of the algorithm, and* $v = |\mathscr{A}\|\mathscr{B}|$. *Then*

$$\Pr\left[|\overline{\mathscr{B}_i}| \geqslant \varepsilon\frac{|\overline{\mathscr{B}}|}{4}\right] \geqslant 1 - \frac{1}{v}. \tag{3}$$

Proof. For $j = 1, \ldots, n$, let $Y_j \stackrel{\text{def}}{=} \mathrm{Est}(\mathscr{B}_j, \mathscr{C})$. Then the random variable $X_j \stackrel{\text{def}}{=} t_2 Y_j/|\mathscr{B}|$ is binomially distributed with parameters t_2 and $|\overline{\mathscr{B}_j}|/|\mathscr{B}|$, and thus by Chernoff bound

$$\Pr[Y_j < \frac{\mathbb{E}[Y_j]}{2}] < \mathrm{e}^{-\mathbb{E}[X_j]/8} \quad \text{for } j = 1, \ldots, n.$$

In particular, for $j{=}k$, we get $\Pr[Y_k < \varepsilon|\overline{\mathscr{B}}|/2] < \mathrm{e}^{-\mathbb{E}[X_k]/8}$ since $\mathbb{E}[Y_k] \geqslant \varepsilon|\overline{\mathscr{B}}|$. Note that, since $\mathrm{Est}(\mathscr{B}, \mathscr{C})$ is the minimum over t_3 independent trials, it follows by Markov inequality that if $|\overline{\mathscr{B}}| < |\mathscr{B}|/4$, then $\Pr[\mathrm{Est}(\mathscr{B}, \mathscr{C}) < |\mathscr{B}|/2] \geqslant 1 - 2^{-t_3}$, and the cleanup step would have already been performed with high probability. On the other hand, if $|\overline{\mathscr{B}}| \geqslant |\mathscr{B}|/4$ then $\mathbb{E}[X_k] = t_2|\overline{\mathscr{B}_k}|/|\mathscr{B}| \geqslant \varepsilon t_2/4$. Thus, it follows that $\Pr[Y_k < \varepsilon|\overline{\mathscr{B}}|/2] < \mathrm{e}^{-\varepsilon t_2/32} + 2^{-t_3}$. Moreover, for any $j \in \mathrm{Ess}(x^*)$ for which $|\overline{\mathscr{B}_j}|/|\overline{\mathscr{B}}| < \varepsilon/4$,we have $\Pr[Y_j \geqslant \varepsilon|\overline{\mathscr{B}}|/2] < 2^{-t_3}$. Consequently,

$$\begin{aligned}&\Pr\left[Y_k \geqslant \frac{\varepsilon|\overline{\mathscr{B}}|}{2} \text{ and } Y_j < \frac{\varepsilon|\overline{\mathscr{B}}|}{2} \text{ for all } j \in \mathrm{Ess}(x^*) \text{ such that } \frac{|\overline{\mathscr{B}_j}|}{|\overline{\mathscr{B}}|} < \frac{\varepsilon}{4}\right]\\ &\quad > 1 - 2^{-t_3}|\mathrm{Ess}(x^*)| - \mathrm{e}^{-\varepsilon t_2/32} \geqslant 1 - \frac{1}{v},\end{aligned}$$

where the last inequality follows by our selection of t_2 and t_3. Since, in Step 6.1, we select the index $i \in [n]$ maximizing Y_i, we have $Y_i \geqslant Y_k$ and thus, with probability at least $1 - 1/v$, we have $|\overline{\mathscr{B}_i}|/|\overline{\mathscr{B}}| \geqslant \varepsilon/4$. □

Lemma 2. *The expected number of recursive calls until a new maximal independent element is output, or procedure* GEN-DUAL($\mathscr{C}$, $\mathscr{A}$, $\mathscr{B}$) *terminates is* $m^{\mathrm{O}(\log^2 m)}$.

Proof. For a node N of the recursion tree $\mathbf{T}$, denote by $\overline{\mathscr{A}} = \overline{\mathscr{A}}(N)$, $\overline{\mathscr{B}} = \overline{\mathscr{B}}(N)$ the subsets of $\mathscr{A}$ and $\mathscr{B}$ intersecting the box specified by node N, and let $v(N) = |\overline{\mathscr{A}}(N)\|\overline{\mathscr{B}}(N)|$. Consider an arbitrary moment during the execution of the algorithm. Let N_0 be the node of $\mathbf{T}$ at which the latest maximal independent element was generated. If no such node exists, then let $N_0 = N_\mathrm{r}$ be the root node of $\mathbf{T}$. For any node N of $\mathbf{T}$, let us denote by $C(v)$, where $v = v(N)$, the expected number of nodes of $\mathbf{T}$ generated (or in other words, the expected total number of recursive calls made) from the moment in time we first visited node N, until we either generate all nodes of the subtree rooted at N, or output a new maximal independent element, whichever happens sooner. Consider any node N of the subtree $\mathbf{T}'$ of $\mathbf{T}$ rooted at N_0. If $s(N) \stackrel{\text{def}}{=} \sum_{a \in \overline{\mathscr{A}}(N)} \frac{1}{2}^{|\mathrm{Ess}(a)|} + \sum_{b \in \overline{\mathscr{B}}(N)} \frac{1}{2}^{|\mathrm{Ess}(b)|} < \frac{1}{2}$, then the probability that the point $z \in \mathscr{C}$, picked randomly in

Step 3 or 4.2 of the procedure, belongs to $\overline{\mathscr{A}}(N)^+ \cup \overline{\mathscr{B}}(N)^-$ is at most $\frac{1}{2}^{t_1}$. Thus, in this case, with probability at least $\sigma_1 \stackrel{\text{def}}{=} 1 - \frac{1}{2}^{t_1}$, we find a new maximal independent element, and with probability at most σ_1 we perform decomposition. In the latter case, we get two subproblems of volumes at most $v-1$ each. This gives the following recurrence for the expected number of recursive calls at node N:

$$C(v) \leqslant 1 + 2(1-\sigma_1)C(v-1) \leqslant 1 + \frac{1}{2^{t_1-1}} C(v-1), \tag{4}$$

where $v = v(N)$. Assume now that $s(N) \geqslant \frac{1}{2}$, let x^* be the element with $|\mathrm{Ess}(x^*)| \leqslant 1/\varepsilon$ found in Step 5, and assume without loss of generality that $x^* \in \mathscr{A}$. Then, by (2), there exists a coordinate $k \in \mathrm{Ess}(x^*)$ such that $|\overline{\mathscr{B}_k}| \geqslant \varepsilon|\overline{\mathscr{B}}|$. By Lemma 1, with probability $p \geqslant \sigma_2 \stackrel{\text{def}}{=} 1 - 1/v$, we can reduce the volume of one of the subproblems, of the current problem, by a factor of at least $1 - \varepsilon/4$. On the other hand, decomposing on the wrong coordinate results in two subproblems of volume at most $v-1$ each, and this can happen with probability $1-p$. Thus for the expected number of recursive calls at node N, we get the following recurrence:

$$\begin{aligned} C(v) &\leqslant p\left[1 + C(v-1) + C\left(\left(1 - \frac{\varepsilon}{4}\right)v\right)\right] + (1-p)[1 + 2C(v-1)] \\ &= 1 + (2-p)C(v-1) + p \cdot C\left(\left(1 - \frac{\varepsilon}{4}\right)v\right) \\ &\leqslant 1 + \left(1 + \frac{1}{v}\right) C(v-1) + C\left(\left(1 - \frac{\varepsilon}{4}\right)v\right). \end{aligned} \tag{5}$$

This recurrence gives $C(v) \leqslant v^{\mathrm{O}(\log^2 v)}$, which is also satisfied by (4). If a maximal independent element is output while traversing the nodes of the subtree $\mathbf{T}'$, then we get the claimed bound on the running time by the above recurrences. Now consider the case when an ancestor node of N_0 in $\mathbf{T}$ has to be visited before either terminating or producing a new maximal independent element. Let N_1 be the node of $\mathbf{T}$ at which the next new maximal independent element is generated. If no such node exists, then let $N_1 = N_r$, the root of the tree $\mathbf{T}$. Let node N_2 in $\mathbf{T}$ be the least common ancestor of N_0 and N_1 in $\mathbf{T}$. Consider the paths $\mathscr{P}(N_0, N_2)$ between nodes N_0 and N_2 and $\mathscr{P}(N_1, N_2)$ between N_1 and N_2 in $\mathbf{T}$. Since a large number of new maximal independent elements may have been added at node N_0, and of course to all its ancestors on the path $\mathscr{P}(N_0, N_2)$, recurrences (4), (5) may no longer hold at the nodes of this path, simply because the branching variable at each such node N was chosen with respect to the old value of $|\mathscr{B}(N)|$ at that node. Assume that the execution order on $\mathbf{T}$ is from left to right, i.e. the algorithm visits a left child of a node before visiting its right child. Since we count the number of new recursive calls made from the time of the last generation that happened at node N_0, each node N on the path $\mathscr{P}(N_0, N_2)$, that has its right child also on this path, does not contribute to this number. Furthermore, the number of recursive calls resulting from the right child N' of each node N on $\mathscr{P}(N_0, N_2)$, that has its left child also on this path, is at most $C(v(N')) \leqslant v(N')^{\mathrm{O}(\log^2 v(N'))}$, as explained above. Since the number of such nodes does not exceed the length of the path $\mathscr{P}(N_0, N_2)$, which is at most m, the expected total number of recursive calls is at most $mC(v)$, where v is the current volume, and the lemma follows. □

We show further that, if $|\mathscr{B}| \gg |\mathscr{A}|$, i.e. if the output size is much bigger than the input size, then the number of recursive calls required for termination, after the last dual element is generated by GEN-DUAL($\mathscr{A}, \mathscr{B}, \mathscr{C}$), is $m^{\mathrm{o}(\log^2 m)}$.

Lemma 3. *Suppose that $\mathscr{A}$ and $\mathscr{B}$ are dual in $\mathscr{C}$, then the expected number of recursive calls until* GEN-DUAL *($\mathscr{C}, \mathscr{A}, \mathscr{B}$) terminates is $m^{\mathrm{O}(\delta \log m)}$, where $m = |\mathscr{A}| + |\mathscr{B}|$ and $\delta = \min\{\log \alpha, \log(\beta/\alpha)/c(\alpha, \beta/\alpha), \log(\alpha/\beta)/c(\beta, \alpha/\beta)\} + 1$, $\alpha = |\mathscr{A}|$, $\beta = |\mathscr{B}|$, and $c = c(a, b)$, is the unique positive root of the equation*

$$2^c(a^{c/\log b} - 1) = 1. \tag{6}$$

Proof. Let $r = \min\{|\mathrm{Ess}(y)| : y \in \mathscr{A} \cup \mathscr{B}\}$, $p = (1 + (\beta/\alpha)^{1/(r-1)})^{-1}$, and let $z \in \mathscr{C}$ be a random element obtained by picking each component independently with $\Pr[z_i = l_i] = p$ and $\Pr[z_i = u_i] = 1 - p$. Then the probability that

$z \in \mathscr{A}^+ \cup \mathscr{B}^-$ is at most $\sum_{a\in\mathscr{A}}(1-p)^{|\mathrm{Ess}(a)|} + \sum_{b\in\mathscr{B}} p^{|\mathrm{Ess}(b)|} \leqslant \alpha(1-p)^r + \beta p^r = \beta p^{r-1}$. Since $\mathscr{A}$ and $\mathscr{B}$ are dual in $\mathscr{C}$, it follows that $\beta p^{r-1} \geqslant 1$, and thus

$$r - 1 \leqslant \frac{\log \beta}{\log(1 + (\beta/\alpha)^{1/(r-1)})}. \tag{7}$$

The maximum value that r can achieve is when both sides of (7) are equal, i.e. r is bounded by the root r' of the equation $\beta^{1/(r'-1)} = 1 + (\beta/\alpha)^{1/(r'-1)}$. If $\alpha = \beta$, then $r' = \log\alpha + 1$. If $\beta > \alpha$, then letting $(\beta/\alpha)^{1/(r'-1)} = 2^c$, we get

$$r' = 1 + \frac{\log(\beta/\alpha)}{c(\alpha, \beta/\alpha)}, \tag{8}$$

where $c(\cdot,\cdot)$ is as defined in (6). The case for $\alpha > \beta$ is similar and the lemma follows from (2) and Lemma 2. □

Note that, if β is much larger than α, then the root r' in (8) is approximately

$$r' \sim 1 + \frac{\log(\beta/\alpha)}{\log(\log(\beta/\alpha)/\log\alpha)}$$

and thus the expected running time of procedure GEN-DUAL($\mathscr{A}, \mathscr{B}, \mathscr{C}$), from the time of last output till termination, is $m^{o(\log^2 m)}$. In fact, one can use Lemma 2 together with the method of conditional expectations to obtain an incremental deterministic algorithm for solving problem GEN($\mathscr{C}, \mathscr{A}$), whose delay between any two successive outputs is $nm^{o(\log^2 m)}$.

3. Joint generation

3.1. 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

$$Ax \geqslant b, \quad x \in \mathscr{C} = \{x \in \mathbb{Z}^n | 0 \leqslant x \leqslant c\}. \tag{9}$$

For $x \in \mathscr{C}$, let $\pi_1(x)$ be the property that x satisfies (9). Then the families $\mathscr{F}_{\pi_1}$ and $\mathscr{G}_{\pi_1}$ correspond, respectively, to the minimal feasible and maximal infeasible vectors for (9). It is known [8] that the family $\mathscr{F}_{\pi_1}$ is (uniformly) dual-bounded:

$$|\mathscr{I}(\mathscr{F}_{\pi_1})| \leqslant rn|\mathscr{F}_{\pi_1}|. \tag{10}$$

Minimal infrequent and maximal frequent sets in binary databases: Let $\mathscr{D} : R \times V \mapsto \{0, 1\}$ 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 $S(X) = S_{\mathscr{D}}(X) \subseteq R$ of all those rows $i \in R$ for which $\mathscr{D}(i, j) = 1$ 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 $|S(X)| \geqslant t$ and otherwise, is said to be *t-infrequent*. For each set $X \in \mathscr{C} \stackrel{\mathrm{def}}{=} 2^V$, let $\pi_2(X)$ be the property that X is t-infrequent. Then π_2 is a monotone property and the families $\mathscr{F}_{\pi_2}$ and $\mathscr{G}_{\pi_2}$ correspond, respectively, to minimal infrequent and maximal frequent sets for $\mathscr{D}$. It is known [11] that

$$|\mathscr{I}(\mathscr{F}_{\pi_2})| \leqslant (r - t + 1)|\mathscr{F}_{\pi_2}|. \tag{11}$$

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

Sparse boxes for multidimensional data: Let $\mathcal{S}$ be a set of points in $\mathbb{R}^n$, and $t \leqslant |\mathcal{S}|$ 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 $\mathcal{B} = \{[a, b] \subseteq \mathbb{R}^n \mid a, b \in \mathcal{C}_1 \times \cdots \times \mathcal{C}_n, a \leqslant b\}$. 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 [9] that the family of maximal t-boxes is (uniformly) dual-bounded:

$$|\mathcal{I}(\mathcal{F}_{\pi_3})| \leqslant |\mathcal{S}||\mathcal{F}_{\pi_3}|. \tag{12}$$

The problem of generating all elements of $\mathcal{F}_{\pi_3}$ has been studied in the machine learning and computational geometry literatures (see [12,14,25]), and is motivated by the discovery of missing associations or "holes" in data mining applications (see [2,23,24]). Edmonds et al. [14] give an algorithm, for solving this problem, whose worst-case time complexity is exponential in the dimension n of the given point set.

3.2. The joint generation algorithm

As before, we assume that we are given an integer box $\mathcal{C}^* = \mathcal{C}_1^* \times \cdots \times \mathcal{C}_n^*$, where $\mathcal{C}_i^* = [l_i^* : u_i^*]$, and $l_i^* \leqslant u_i^*$, 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 [8], for which an efficient implementation was given in the previous section. Again, 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. The same decomposition lemma, stated in Section 2.1, holds also here. More precisely, either (i) there is an element $x \in \mathcal{F} \cup \mathcal{G}$ with at most $1/\varepsilon$ essential coordinates, where $\varepsilon \stackrel{\text{def}}{=} 1/(1 + \log m)$ 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 $i = 1, \ldots, n$. As before, in case (i), one can decompose the problem into two strictly smaller subproblems, and the algorithm is guaranteed to reduce the cardinality of one of the sets $\mathcal{F}$ or $\mathcal{G}$ by a factor of at least $1 - \varepsilon$ at each recursive step.

We used the following data structures in our implementation:

- Two global 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 $\mathcal{G}^*$, that appear in the current subproblem.
- Two global balanced binary search trees $\mathbf{H}(\mathcal{F}^*)$ and $\mathbf{H}(\mathcal{G}^*)$, built on the elements of $\mathcal{F}^*$ and $\mathcal{G}^*$, respectively, using lexicographic ordering. Each node of the tree $\mathbf{H}(\mathcal{F}^*)$ ($\mathbf{H}(\mathcal{G}^*)$) 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.

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

Minimization $\min_{\mathcal{F}_\pi}(z)$: It takes as input a vector $z \in \mathcal{F}_\pi^+$ and returns a minimal vector z^* in $\mathcal{F}_\pi^+ \cap \{z\}^-$. As explained in the proof of Proposition 1, such a vector $z^* = \min_{\mathcal{F}_\pi}(z)$ can be computed by coordinate descent. 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 $a_j x \geqslant b_j$ 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

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

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

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

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

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

Exhaustive duality $(\mathscr{C}, \pi, \mathscr{F}, \mathscr{G})$: Assuming $|\mathscr{F}\|\mathscr{G}| \leqslant 1$,check if there are no other vectors in $\mathscr{C}\backslash(\mathscr{F}^+ \cup \mathscr{G}^-)$ as follows. First,if $|\mathscr{F}| = |\mathscr{G}| = 1$ then,as explained in Section 2.2,we either can find a vector $z \notin \mathscr{F}^+ \cup \mathscr{G}^-$,in which case return either $\min_{\mathscr{F}_\pi}(z)$ or $\max_{\mathscr{G}_\pi}(z)$ depending on whether $\pi(z) = 1$ or $\pi(z) = 0$,respectively,or we declare that $\mathscr{F}$ and $\mathscr{G}$ are dual in $\mathscr{C}$. Second,if $|\mathscr{F}| = 0$ then we check satisfiability of u. If $\pi(u) = 1$ then return $\min_{\mathscr{F}_\pi}(u)$. Else,if $\pi(u) = 0$ then let $z = \max_{\mathscr{G}_\pi}(u)$,and return either Λ or z depending on whether $z \in \mathscr{G}^*$ or not (this check can be done in $O(n \log |\mathscr{G}^*|)$ using the search tree $\mathbf{H}(\mathscr{G}^*)$). Finally,if $|\mathscr{G}| = 0$ then check satisfiability of l. If $\pi(l) = 0$ then return $\max_{\mathscr{G}_\pi}(l)$. Else,if $\pi(l) = 1$ then let $z = \min_{\mathscr{F}_\pi}(l)$,and return either Λ or z depending on whether $z \in \mathscr{F}^*$ or not. This step takes $O(\max\{n \log |\mathscr{F}^*| + T_{\min}, n \log |\mathscr{G}^*| + T_{\max}\})$ time.

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

Count estimation: For a subset $\mathscr{X} \subseteq \mathscr{F}$ (or $\mathscr{X} \subseteq \mathscr{G}$), use sampling to estimate the number $\mathrm{Est}(\mathscr{X}, \mathscr{C})$ of elements of $\mathscr{X} \subseteq \mathscr{F}$ (or $\mathscr{X} \subseteq \mathscr{G}$) that are active with respect to the current box $\mathscr{C}$. This can be done as explained in the similar procedure in Section 2.2.

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

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

The following result, regarding the expected running time of the algorithm, is inherent from Lemma 2.

Proposition 3. *The expected number of recursive calls until a new element in* $(\mathscr{F}_\pi \backslash \mathscr{F}^*) \cup (\mathscr{G}_\pi \backslash \mathscr{G}^*)$ *is output, or procedure* GEN$(\mathscr{C}, \pi, \mathscr{F}, \mathscr{G})$ *terminates is* $m^{O(\log^2 m)}$.

However, as we shall see from the experiments, the algorithm seems to practically behave much more efficiently than indicated by Proposition 3. In fact, in most of the experiments we performed, we got an almost average linear delay (in m) for generating a new point in $(\mathscr{F}_\pi \backslash \mathscr{F}^*) \cup (\mathscr{G}_\pi \backslash \mathscr{G}^*)$.

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

$\mathscr{F}^o \leftarrow \emptyset, \mathscr{G}^o \leftarrow \emptyset$.
While $|\mathscr{F}\|\mathscr{G}| \leqslant 1$
  $z \leftarrow$ Exhaustive duality$\left(\mathscr{C}, \pi, \mathscr{F}, \mathscr{G}\right)$.
  If $z = \Lambda$ then return$\left(\mathscr{F}^o, \mathscr{G}^o\right)$.
  else if $\pi(z) = 1$ then $\mathscr{F} \leftarrow \mathscr{F} \cup \{z\}, \mathscr{F}^o \leftarrow \mathscr{F}^o \cup \{z\}, \mathscr{F}^* \leftarrow \mathscr{F}^* \cup \{z\}$.
  else $\mathscr{G} \leftarrow \mathscr{G} \cup \{z\}, \mathscr{G}^o \leftarrow \mathscr{G}^o \cup \{z\}, \mathscr{G}^* \leftarrow \mathscr{G}^* \cup \{z\}$.
end while
$z \leftarrow$ Random solution$\left(\mathscr{C}, \pi, \mathscr{F}^*, \mathscr{G}^*\right)$.
While $(z \neq \Lambda)$ do
  if $\pi(z) = 1$ then $\mathscr{F} \leftarrow \mathscr{F} \cup \{z\}, \mathscr{F}^o \leftarrow \mathscr{F}^o \cup \{z\}, \mathscr{F}^* \leftarrow \mathscr{F}^* \cup \{z\}$.
  else $\mathscr{G} \leftarrow \mathscr{G} \cup \{z\}, \mathscr{G}^o \leftarrow \mathscr{G}^o \cup \{z\}, \mathscr{G}^* \leftarrow \mathscr{G}^* \cup \{z\}$.
  $z \leftarrow$ Random Solution$\left(\mathscr{C}, \pi, \mathscr{F}^*, \mathscr{G}^*\right)$.
end while
$x^* \leftarrow \text{argmin}\left\{|\text{Ess}(y)| : y \in \left(\mathscr{F} \cap \mathscr{C}^-\right) \cup \left(\mathscr{G} \cap \mathscr{C}^+\right)\right\}$.
If $x^* \in \mathscr{F}$ then
  $i \leftarrow \text{argmax}\{\text{Est}(\{b \in \mathscr{G} : b_j < x^*_j\}, \mathscr{C}) : j \in \text{Ess}(x^*)\}$.
  $\mathscr{C}' = \mathscr{C}_1 \times \cdots \times \mathscr{C}_{i-1} \times [x^*_i : u_i] \times \mathscr{C}_{i+1} \times \cdots \times \mathscr{C}_n$.
  If Est $\left(\mathscr{G}, \mathscr{C}'\right) \leqslant f * |\mathscr{G}|$ then
    $\mathscr{G}' \leftarrow$ Cleanup$(\mathscr{G}, \mathscr{C}')$.
  else
    $\mathscr{G}' \leftarrow \mathscr{G}$.
  $\left(\mathscr{F}_l, \mathscr{G}_l\right) \leftarrow$ GEN $\left(\mathscr{C}', \pi, \mathscr{F}, \mathscr{G}'\right)$.
  $\mathscr{F}^o \leftarrow \mathscr{F}^o \cup \mathscr{F}_l, \mathscr{F} \leftarrow \mathscr{F} \cup \mathscr{F}_l, \mathscr{F}^* \leftarrow \mathscr{F}^* \cup \mathscr{F}_l$.
  $\mathscr{G}^o \leftarrow \mathscr{G}^o \cup \mathscr{G}_l, \mathscr{G} \leftarrow \mathscr{G} \cup \mathscr{G}_l, \mathscr{G}^* \leftarrow \mathscr{G}^* \cup \mathscr{G}_l$.
  $\mathscr{C}'' = \mathscr{C}_1 \times \cdots \times \mathscr{C}_{i-1} \times [l_i : x^*_i - 1] \times \mathscr{C}_{i+1} \times \cdots \times \mathscr{C}_n$.
  If Est $\left(\mathscr{F}, \mathscr{C}''\right) \leqslant f * |\mathscr{F}|$ then
    $\mathscr{F}'' \leftarrow$ Cleanup$\left(\mathscr{F}, \mathscr{C}''\right)$.
  else
    $\mathscr{F}'' \leftarrow \mathscr{F}$.
  $\left(\mathscr{F}_r, \mathscr{G}_r\right) \leftarrow$ GEN $\left(\mathscr{C}'', \pi, \mathscr{F}'', \mathscr{G}\right)$.
  $\mathscr{F}^o \leftarrow \mathscr{F}^o \cup \mathscr{F}_r, \mathscr{F}^* \leftarrow \mathscr{F}^* \cup \mathscr{F}_r, \mathscr{G}^o \leftarrow \mathscr{G}^o \cup \mathscr{G}_r, \mathscr{G}^* \leftarrow \mathscr{G}^* \cup \mathscr{G}_r$.
else
  Symmetric versions for Steps 16–31 above (details omitted).
end if
Return $\left(\mathscr{F}^o, \mathscr{G}^o\right)$.

4. Experimental results

4.1. Hypergraph transversals

We performed a number of experiments to evaluate our implementation of the hypergraph transversal algorithm. Six types of hypergraphs were used in the experiments:

- Random (denoted henceforth by $R(n, \alpha, d)$): this is a hypergraph with α hyperedges, each of which is picked randomly by first selecting its size k uniformly from $[2 : d]$ and then randomly selecting k elements of $[n]$ (in fact, in some experiments, we fix $k = d$ for all hyperedges).
- Matching ($M(n)$): this is a graph on n vertices (n is even) with $n/2$ edges forming an induced matching.
- Matching dual ($MD(n)$): this is just $M(n)^{\mathrm{d}}$, the transversal hypergraph of $M(n)$. In particular, it has $2^{n/2}$ hyperedges on n vertices.
- Threshold graph ($TH(n)$): this is a graph on n vertices numbered from 1 to n (where n is even), with edge set $\{\{i, j\} : 1 \leqslant i < j \leqslant n, j \text{ is even}\}$ (i.e. for $j = 2, 4, \ldots, n$, there is an edge between i and j for all $i < j$). The reason

Table 1
Performance of the algorithm for the class $R(n, \alpha, d)$ of hypergraphs, where $2 \leqslant d \leqslant n-1$

n	30			50			60		
$\alpha = \lvert\mathscr{H}\rvert$	275	213	114	507	441	342	731	594	520
Total time (s)	0.1	0.3	3.1	43.3	165.6	1746.8	322.2	2220.4	13,329.5
Output size $\lvert\mathscr{H}^{\mathrm{d}}\rvert$	150	450	5.7×10^3	1.7×10^4	6.4×10^4	4.7×10^5	7.5×10^4	4.7×10^5	1.7×10^6
Time/trans. (ms)	0.7	0.7	0.5	2.5	2.6	3.7	4.3	4.7	7.8

Table 2
Performance of the algorithm for the induced matching $M(n)$

n	20	22	24	26	28	30	32	34	36	38	40
$\alpha = \lvert\mathscr{H}\rvert$	10	11	12	13	14	15	16	17	18	19	20
Total time (s)	0.3	0.7	1.4	3.2	7.1	17.8	33.9	80.9	177.5	418.2	813.1
Output size $\lvert\mathscr{H}^{\mathrm{d}}\rvert$	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Time/trans. (ms)	0.3	0.3	0.3	0.4	0.4	0.5	0.5	0.6	0.7	0.8	0.8

Table 3
Performance of the algorithm for the matching dual $MD(n)$

n	20	22	24	26	28	30	32	34	36	38	40
$\alpha = \lvert\mathscr{H}\rvert$	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Total time (s)	0.15	0.44	1.3	4.28	13.3	42.2	132.7	421.0	1330.3	4377.3	14,010.5
Output size $\lvert\mathscr{H}^{d}\rvert$	10	11	12	13	14	15	16	17	18	19	20
Time/trans. (ms)	15	40	108	329	950	2.8×10^3	8.2×10^3	2.5×10^4	7.4×10^4	2.3×10^5	7.0×10^5

Table 4
Performance of the algorithm for the threshold graph $TH(n)$

n	20	40	60	80	100	120	140	160	180	200
$\alpha = \lvert\mathscr{H}\rvert$	100	400	900	1600	2500	3600	4900	6400	8100	10,000
Total time (s)	0.02	0.4	1.9	6.0	18.4	40.2	78.2	142.2	232.5	365.0
Output size $\lvert\mathscr{H}^{d}\rvert$	11	21	31	41	51	61	71	81	91	101
Time/trans. (ms)	1.8	33.3	146.2	428.6	1.2×10^3	2.5×10^3	4.6×10^3	7.9×10^3	12.2×10^3	18.3×10^3

we are interested in such kind of graphs is that they are known to have both a small number of edges (namely, $n^2/4$) and a small number of transversals (namely, $n/2+1$ for even n).

- Self-dualized threshold graph ($SDTH(n)$): this is a self-dual hypergraph $\mathscr{H}$ on n vertices obtained from the threshold graph and its dual $TH(n-2)$, $TH(n-2)^{\mathrm{d}} \subseteq 2^{[n-2]}$ as follows:

$$\mathscr{H} = \{\{n-1, n\}\} \cup \{\{n-1\} \cup H \mid H \in TH(n-2)\} \cup \{\{n\} \cup H \mid H \in TH(n-2)^{\mathrm{d}}\}.$$

  This gives a family of hypergraphs with polynomially bounded input and output sizes $|SDTH(n)| = |SDTH(n)^{\mathrm{d}}| = (n-2)^2/4 + n/2 + 1$.
- Self-dualized Fano-plane product ($SDFP(n)$): this is constructed by starting with the hypergraph $\mathscr{H}_0 = \{\{1,2,3\}, \{1,5,6\}, \{1,7,4\}, \{2,4,5\}, \{2,6,7\}, \{3,4,6\}, \{3,5,7\}\}$ (which represents the set of lines in a Fano plane and is self-dual), taking $k = (n-2)/7$ disjoint copies $\mathscr{H}_1, \ldots, \mathscr{H}_k$ of $\mathscr{H}_0$, and letting $\mathscr{H} = \mathscr{H}_1 \cup \cdots \cup \mathscr{H}_k$. The dual hypergraph $\mathscr{H}^{\mathrm{d}}$ is just the hypergraph of all 7^k unions obtained by taking one hyperedge from each of the hypergraphs $\mathscr{H}_1, \ldots, \mathscr{H}_k$. Finally, we define the hypergraph $SDFP(k)$ to be the hypergraph of $1 + 7k + 7^k$ hyperedges on n vertices, obtained by self-dualizing $\mathscr{H}$, as we did for threshold graphs.

The experiments were performed on a *Pentium* 4 processor with 2.2 GHz of speed and 512 MB of memory. Tables 1–6 summarize our results for several instances of the different classes of hypergraphs listed above. In the tables, for the specified hypergraphs with the specified parameters, we show: (i) the total CPU time, in seconds, required to generate

Table 5
Performance of the algorithm for the class $SDTH(n)$

n	22	42	62	82	102	122	142	162	182	202
$\alpha = \|\mathscr{H}\|$	112	422	932	1642	2552	3662	4972	6482	8192	10,102
Total time (s)	0.07	0.9	5.9	23.2	104.0	388.3	1164.2	2634.0	4820.6	8720.0
Output size $\|\mathscr{H}^d\|$	112	422	932	1642	2552	3662	4972	6482	8192	10,102
Time/trans. (ms)	0.6	2.1	6.3	14.1	40.8	106	234	406	588	863

Table 6
Performance of the algorithm for the class $SDFP(n)$

n	9	16	23	30	37
$\alpha = \|\mathscr{H}\|$	15	64	365	2430	16,843
Total time (s)	0.01	0.1	4.8	198.1	11,885.1
Output size $\|\mathscr{H}^{\mathrm{d}}\|$	15	64	365	2430	16843
Time/trans. (ms)	0.7	1.6	13.2	81.5	706

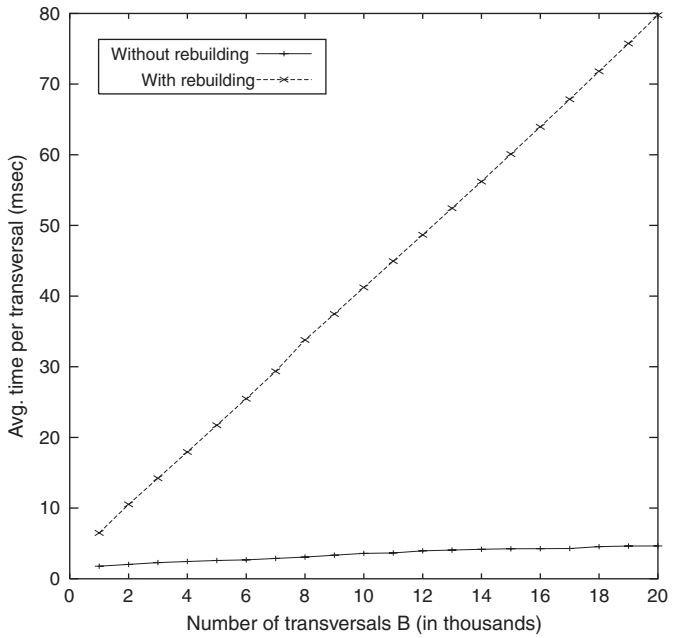

Fig. 1. Effect of rebuilding the recursion tree. Each plot shows the average CPU time (in milliseconds) per generated transversal versus the number of transversals, for hypergraphs of type $R(30, 100, 5)$.

all transversals, (ii) the total size of the transversal hypergraph, and (iii) the average time per generated transversal. For random hypergraphs, the time and output size reported are the averages over 30 experiments. For random hypergraphs, we only show results for $n \leqslant 60$ vertices. For larger numbers of vertices, the number of transversals becomes very large (although the delay between successive transversals is still acceptable).

We also performed some experiments to compare different implementations of the algorithm and to study the effect of increasing the number of vertices and the number of hyperedges on the performance. In particular, Fig. 1 shows the effect of rebuilding the tree each time a transversal is generated on the output rate. From this figure we see that the average time per transversal is almost constant if we do not rebuild the tree. In Fig. 2, we show that the randomized implementation of the algorithm offers substantial improvement over the deterministic one. Figs. 3 and 4, respectively,

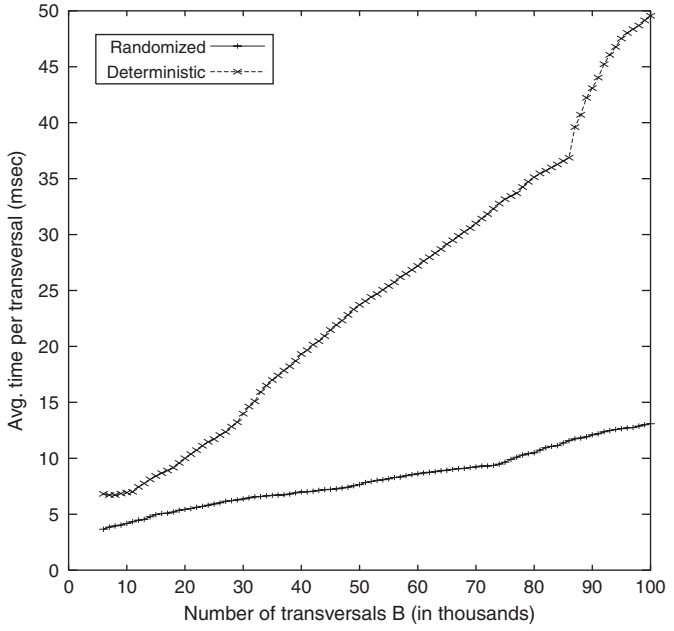

Fig. 2. Comparing deterministic versus randomized implementations. Each plot shows the average CPU time/transversal versus the number of transversals, for hypergraphs of type $R(50, 100, 10)$.

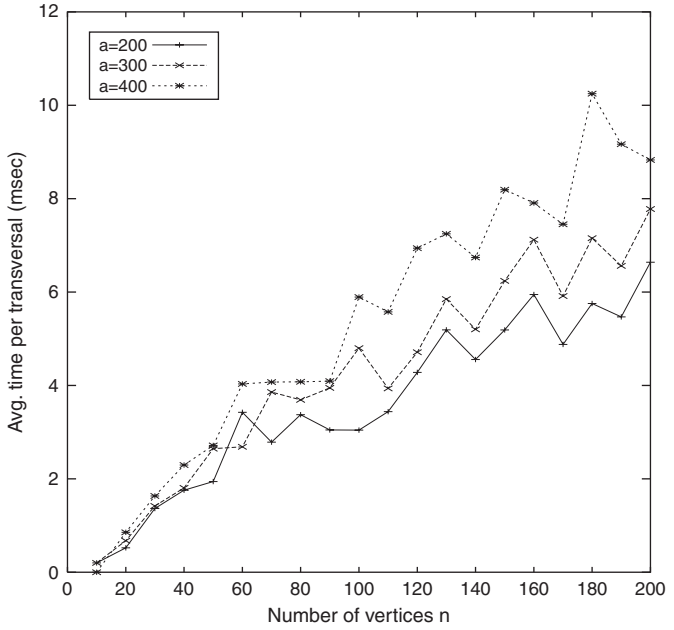

Fig. 3. Average CPU time/transversal versus the number of vertices n for random hypergraphs $R(n, a, d)$, where $d = n/4$, and $a = 200, 300, 400$.

show how the average CPU time/transversal changes as the number of vertices n and the number of hyperedges α are increased. The plots show that the average CPU time/transversal does not increase more than linearly with increasing α or n.

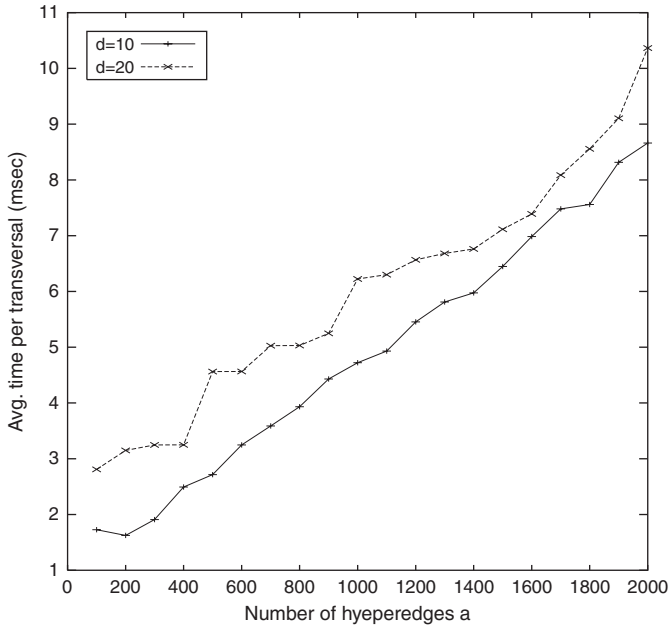

Fig. 4. Average CPU time/transversal versus the number of hyperedges a for random hypergraphs $R(50, a, d)$, for $d = 10, 20$.

Table 7
Performance of the algorithm for property π_1, where $r = 5$ and $c = 2$

n	10		20		30		40		50	
	$\mathscr{F}_{\pi_1}$	$\mathscr{G}_{\pi_1}$	$\mathscr{F}_{\pi_1}$	$\mathscr{G}_{\pi_1}$	$\mathscr{F}_{\pi_1}$	$\mathscr{G}_{\pi_1}$	$\mathscr{F}_{\pi_1}$	$\mathscr{G}_{\pi_1}$	$\mathscr{F}_{\pi_1}$	$\mathscr{G}_{\pi_1}$
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 (s)	4.7	4.7	297	297	1627	1625	5844	5753	10,703	10,700
Time/output (ms)	13	24	27	62	29	78	40	103	50	133
Ratio $\lvert\mathscr{G}_{\pi_1}\rvert/\lvert\mathscr{F}_{\pi_1}\rvert$	0.60		0.57		0.40		0.47		0.38	

4.2. Joint generation

We performed a number of experiments to evaluate our joint generation implementation on random instances of the three monotone properties described in Section 3.1. The experiments were also performed on a *Pentium 4* processor with 2.2 GHz of speed and 512 MB 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 $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$ 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 7 and 8 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 $\mathscr{F}_{\pi_1}$ to the size of $\mathscr{G}_{\pi_1}$ for comparison with the worst-case bound of (10). Note that this ratio is relatively close to 1, making joint generation an efficient method for generating both families $\mathscr{F}_{\pi_1}$ and $\mathscr{G}_{\pi_1}$.

Tables 9 and 10 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 9, for $t = 1, 2$, the bias between the numbers of maximal frequent

Table 8
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 (s)	408	408	2244	2242	6495	6482	15,857	15,856	30,170	30,156
Time/output (ms)	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	

Table 9
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 (s)	60	55	1520	769	22,820	3413	1962	1942	13,269	13,214	28,824	28,737
Time/output (ms)	20	690	30	7742	68	34,184	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 10
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 (s)	7534	7523	6511	6508	4349	4346	3031	3029	13,895	13,890	9896	9890
Time/output (ms)	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	

sets and minimal infrequent sets, for the shown random examples, seems 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 10 illustrates this on a number of examples in which larger values of the threshold were used.

Figs. 5 and 6 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 efficiency decreases (the generation problem for maximal infeasible solutions is NP-hard). For minimal infrequent and maximal frequent sets, Fig. 6 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 3.

Table 11 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. We are not aware of any implementation of an algorithm for generating maximal sparse boxes except for [14] which presents some experiments for $n = 2$ and $t = 0$. Experiments in [14] indicated that the algorithm suggested there is almost linear in the number of points r. Fig. 7 illustrates a similar behavior exhibited by our algorithm. In the figure,

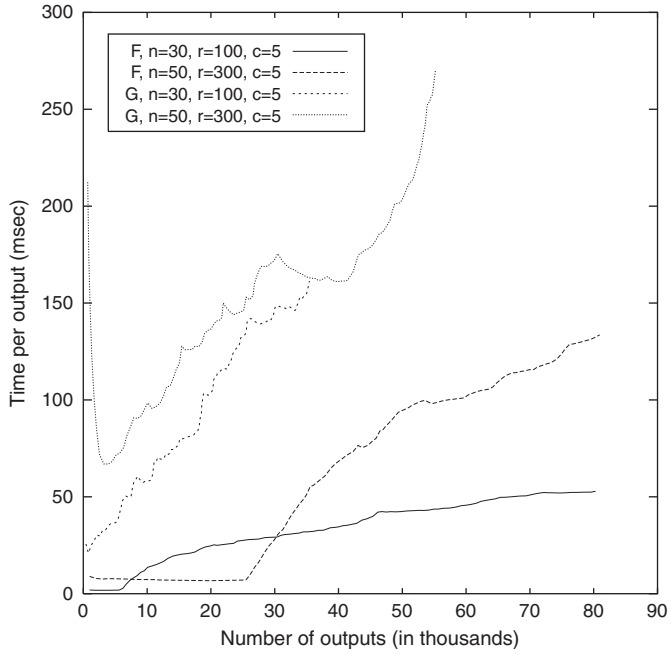

Fig. 5. Average time per output, as a function of the number of outputs for minimal feasible/maximal infeasible solutions of linear systems, with $c = 5$ and $(n, r) = (30, 100), (50, 300)$.

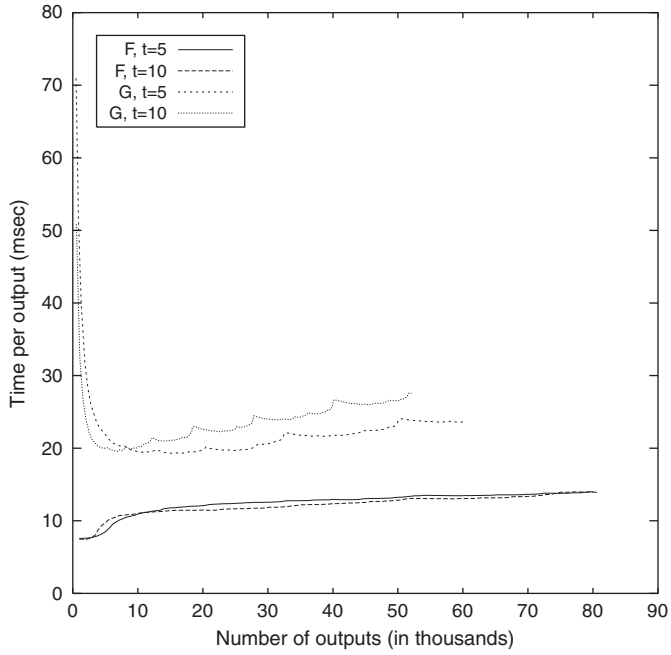

Fig. 6. Average time per output, as a function of the number of outputs for minimal infrequent/maximal frequent sets, with $n = 30$, $r = 1000$ and $t = 5, 10$.

Table 11
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	
	$\mathscr{F}_{\pi_3}$	$\mathscr{G}_{\pi_3}$	$\mathscr{F}_{\pi_3}$	$\mathscr{G}_{\pi_3}$	$\mathscr{F}_{\pi_3}$	$\mathscr{G}_{\pi_3}$	$\mathscr{F}_{\pi_3}$	$\mathscr{G}_{\pi_3}$	$\mathscr{F}_{\pi_3}$	$\mathscr{G}_{\pi_3}$	$\mathscr{F}_{\pi_3}$	$\mathscr{G}_{\pi_3}$
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 (s)	932	623	2658	1456	3924	2933	8731	8724	17,408	17,404	16,156	16,156
Time/output (ms)	29	6237	27	4866	27	5889	19	98	23	37	24	35
Ratio $\lvert\mathscr{G}_{\pi_3}\rvert/\lvert\mathscr{F}_{\pi_3}\rvert$	0.0062		0.0061		0.0068		0.3881		1.2488		1.3818	

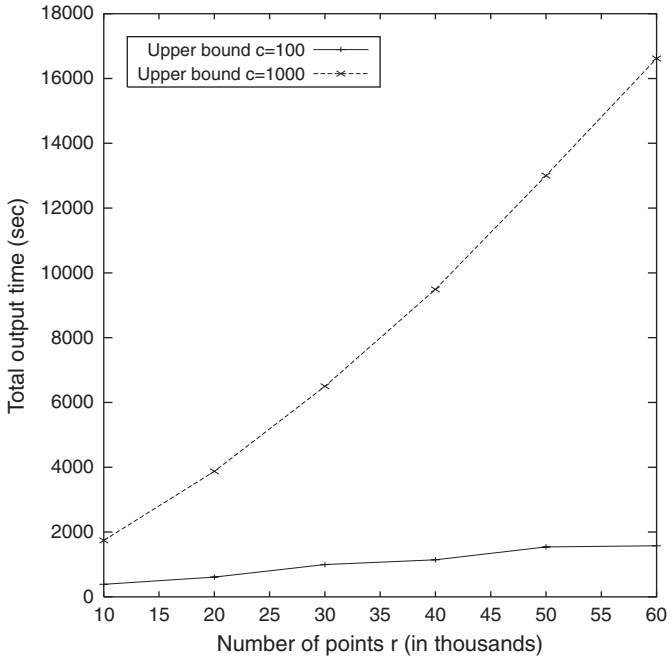

Fig. 7. Total generation time as a function of the number of points r for maximal boxes with $n = 2$, $t = 0$, and $c = 100$, 1000.

we show the total time required to generate all the two-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.

As mentioned in Section 3.2, it is possible in general to implement the procedures $\min_{\mathscr{F}}(z)$ and $\max_{\mathscr{G}}(z)$ using the coordinate descent method, but more efficient implementations can be obtained if we specialize these procedures to the monotone property under consideration. Fig. 8 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 $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$ become closer in size. This observation is illustrated in Fig. 9 which plots the average time per output (i.e. total time to output all the elements of $\mathscr{F}_\pi \cup \mathscr{G}_\pi$ divided by $|\mathscr{F}_\pi \cup \mathscr{G}_\pi|$) versus the ratio $|\mathscr{G}_\pi|/|\mathscr{F}_\pi|$. This indicates that, when the elements of the sets $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$ 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 $\mathscr{F}^* \subseteq \mathscr{F}_\pi$ and $\mathscr{G}^* \subseteq \mathscr{G}_\pi$.

5. Conclusion

We have presented an efficient implementation of an algorithm for generating maximal independent elements for a family of vectors in an integer box. Experiments show that this implementation performs well in practice. We are not aware of any experimental evaluation of algorithms for generating hypergraph transversals except for [21,4] in which

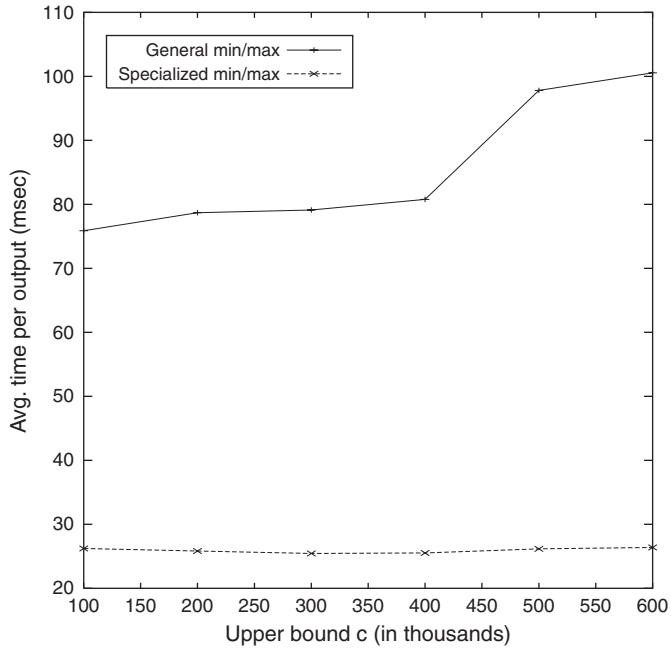

Fig. 8. Comparing general versus specialized minimization for property π_3. Each plot shows the average CPU time/maximal empty box generated versus the upper bound c, for $n = 5$ and $r = 500$.

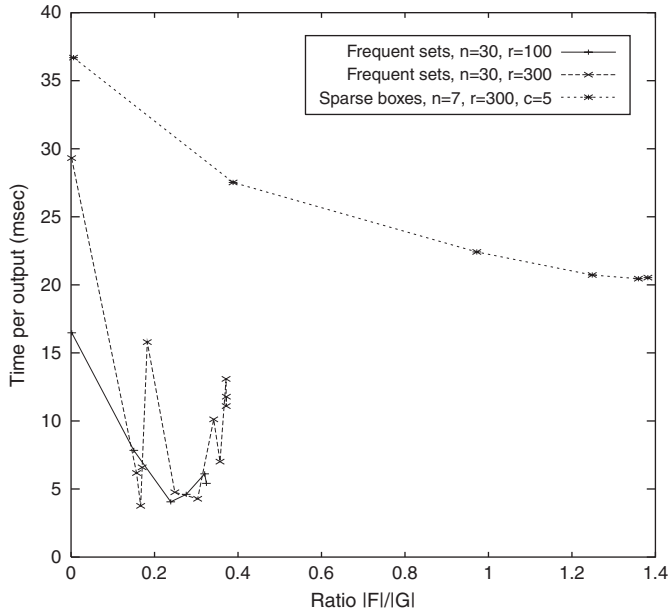

Fig. 9. Average generation time as a function of the ratio $|\mathscr{G}_\pi|/|\mathscr{F}_\pi|$, for properties π_2 and π_3.

heuristics for solving this problem were described and experimentally evaluated. However, the results in [21] show the performance for relatively small instances which are easy cases for our implementation. On the other hand, the method described in this paper can handle much larger instances due to the fact that it scales nicely with the size of the problem. In particular, our code can produce, in a few hours, millions of transversals even for hypergraphs with hundreds of

Table 12
Comparing the performance of our transversal algorithm with that of [4]

n	62	82	102	122	142	162
Our implementation	20.7	89.1	408.9	1022.5	2280.3	3140.1
[4]	17.9	105.6	514.5	1516.4	3792.0	8656.0

The table shows the total CPU time in seconds required to generate all transversals for hypergraphs in the class $SDTH(n)$.

vertices and thousands of hyperedges. Furthermore, the experiments also indicate that the delay per transversal scales almost linearly with the number of vertices and number of hyperedges. As for the code in [4], we observed that, although this code seems to perform very well in the case of random hypergraphs, it increasingly slows down with increasing the number of vertices for non-random instances such as self-dualized threshold graphs (see Table 12 for comparison with our implementation for that class of hypergraphs).

We have also presented an efficient implementation for a quasi-polynomial algorithm for jointly generating the families $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$ 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 $\mathscr{F}_\pi$ and $\mathscr{G}_\pi$ 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 (10)–(12) may not hold in general.

Acknowledgments

We thank Thomas Manoukian for providing us with the source code and the reference for [4].

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining associations between sets of items in massive databases, Proceedings of the 1993 ACM-SIGMOD International Conference, 1993, pp. 207–216.
[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo, Fast discovery of association rules, in: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), Advances in Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, CA, 1996, pp. 307–328.
[3] M. Anthony, N. Biggs, Computational Learning Theory, Cambridge University Press, Cambridge, 1992.
[4] J. Bailey, T. Manoukian, K. Ramamohanarao, A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns, in: The Proceedings of the Third IEEE International Conference on Data Mining (ICDM), Florida, USA, November 2003, pp. 485–488.
[5] J.C. Bioch, T. Ibaraki, Complexity of identification and dualization of positive Boolean functions, Inform. and Comput. 123 (1995) 50–63.
[6] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, Generating dual-bounded hypergraphs, Optimization Methods and Software (OMS)—Part I 17 (5) (2002) 749–781.
[7] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, Enumerating minimal dicuts and strongly connected subgraphs and related geometric problems, in: D. Bienstock, G. Nemhauser (Eds.), The 10th Conference on Integer Programming and Combinatorial Optimization (IPCO X), Lecture Notes in Computer Science, vol. 3064, June 2004, pp. 152–162.
[8] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, K. Makino, Dual-bounded generating problems: all minimal integer solutions for a monotone system of linear inequalities, SIAM J. Comput. 31 (5) (2002) 1624–1643.
[9] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, K. Makino, An intersection inequality for discrete distributions and related generation problems, in: Automata, Languages and Programming, 30th International Colloqium, ICALP 2003, Lecture Notes in Computer Science, vol. 2719, 2003, pp. 543–555.
[10] E. Boros, V. Gurvich, L. Khachiyan, K. Makino, Dual-bounded generating problems: partial and multiple transversals of a hypergraph, SIAM J. Comput. 30 (6) (2001) 2036–2050.
[11] E. Boros, V. Gurvich, L. Khachiyan, K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets, in: 19th International Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science, vol. 2285, March 2002, pp. 133–141.
[12] B. Chazelle, R.L. (Scot) Drysdale III, D.T. Lee, Computing the largest empty rectangle, SIAM J. Comput. 15 (1) (1986) 550–555.
[13] C.J. Colbourn, The Combinatorics of Network Reliability, Oxford University Press, OXford, 1987.
[14] J. Edmonds, J. Gryz, D. Liang, R.J. Miller, Mining for empty rectangles in large data sets, in: Proceedings of the Eighth International Conference on Database Theory (ICDT), Lecture Notes in Computer Science, vol. 1973, January 2001, pp. 174–188.
[15] T. Eiter, G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems, SIAM J. Comput. 24 (1995) 1278–1304.
[16] M.L. Fredman, L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, J. Algorithms 21 (1996) 618–628.
[17] D. Gunopulos, R. Khardon, H. Mannila, H. Toivonen, Data mining, hypergraph transversals and machine learning, in: Proceedings of the 16th ACM-PODS Conference, 1997, pp. 209–216.

[18] V. Gurvich, To theory of multistep games, USSR Comput. Math. and Math Phys. 13–6 (1973) 1485–1500.
[19] V. Gurvich, Nash-solvability of games in pure strategies, USSR Comput. Math. and Math. Phys. 15 (1975) 357–371.
[20] V. Gurvich, L. Khachiyan, On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions, Discrete Appl. Math. 96–97 (1999) 363–373.
[21] D.J. Kavvadias, E.C. Stavropoulos, Evaluation of an algorithm for the transversal hypergraph problem, in: Proceedings of the Third Workshop on Algorithm Engineering (WAE'99), Lecture Notes in Computer Science, vol. 1668, 1999, pp. 72–84.
[22] E. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, SIAM J. Comput. 9 (1980) 558–565.
[23] B. Liu, L.-P. Ku, W. Hsu, Discovering interesting holes in data, in: Proceedings of the IJCAI, Nagoya, Japan, 1997, pp. 930–935.
[24] B. Liu, K. Wang, L.-F. Mun, X.-Z. Qi, Using decision tree induction for discovering holes in data, in: Proceedings of the Fifth Pacific Rim International Conference on Artificial Intelligence, 1998, pp. 182–193.
[25] M. Orlowski, A new algorithm for the large empty rectangle problem, Algorithmica 5 (1) (1990) 65–73.
[26] R.C. Read, Every one a winner, or how to avoid isomorphism when cataloging combinatorial configurations, Ann. Discrete Math. 2 (1978) 107–120.
[27] R. Srikant, R. Agrawal, Mining quantitative association rules in large relational tables, in: Proceedings of the ACM-SIGMOD 1996 International Conference on Management of Data, Montreal, Canada, June 1996, pp. 1–12.