An Efficient Implementation of a Quasi-polynomial Algorithm for Generating Hypergraph Transversals*

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

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003; {boros,elbassio,gurvich}@rutcor.rutgers.edu

[2] Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway NJ 08854-8003; leonid@cs.rutgers.edu

Abstract. Given a finite set V, and a hypergraph $\mathcal{H} \subseteq 2^V$, the hypergraph transversal problem calls for enumerating all minimal hitting sets (transversals) for $\mathcal{H}$. This problem plays an important role in practical applications as many other problems were shown to be polynomially equivalent to it. Fredman and Khachiyan (1996) gave an incremental quasi-polynomial time algorithm for solving the hypergraph transversal problem [9]. In this paper, we present an efficient implementation of this algorithm. While we show that our implementation achieves the same bound on the running time as in [9], practical experience with this implementation shows that it can be substantially faster. We also show that a slight modification of the algorithm in [9] can be used to give a stronger bound on the running time.

1 Introduction

Let V be a finite set of cardinality $|V| = n$. For a hypergraph $\mathcal{H} \subseteq 2^V$, let us denote by $\mathcal{I}(\mathcal{H})$ the family of its *maximal independent* sets, i.e. maximal subsets of V not containing any hyperedge of $\mathcal{H}$. The complement of a maximal independent subset is called a *minimal transversal* of $\mathcal{H}$ (i.e. minimal subset of V intersecting all hyperedges of $\mathcal{H}$). The collection $\mathcal{H}^d$ of minimal transversals is also called the *dual* or *transversal* hypergraph for $\mathcal{H}$. The *hypergraph transversal problem* is the problem of generating all transversals of a given hypergraph. This problem has important applications in combinatorics [14], artificial intelligence [8], game theory [11,12], reliability theory [7], database theory [6,8,10], integer programming [3], learning theory [1], and data mining [2,5,6].

The *theoretically* best known algorithm for solving the hypergraph transversal problem is due to Fredman and Khachiyan [9] and works by performing $|\mathcal{H}^d| + 1$ calls to the following problem, known as *hypergraph dualization*:

* This research was supported by the National Science Foundation (Grant IIS-0118635), and by the Office of Naval Research (Grant N00014-92-J-1375). The third author is also grateful for the partial support by DIMACS, the National Science Foundation's Center for Discrete Mathematics and Theoretical Computer Science.

G. Di Battista and U. Zwick (Eds.): ESA 2003, LNCS 2832, pp. 556–567, 2003.

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

Two recursive algorithms were proposed in [9] to solve the hypergraph dualization problem. These algorithms have incremental quasi-polynomial time complexities of $poly(n) + m^{O(\log^2 m)}$ and $poly(n) + m^{o(\log m)}$ respectively, where $m = |\mathcal{H}| + |\mathcal{X}|$. 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 [9], 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 [3]. In this latter problem, we are given an integral box $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$, where $\mathcal{C}_i$ is a finite set of consecutive integers, and a subset $\mathcal{A} \subseteq \mathcal{C}$. Denote by $\mathcal{A}^+ = \{x \in \mathcal{C} \mid x \geq a, \text{ for some } a \in \mathcal{A}\}$ and $\mathcal{A}^- = \{x \in \mathcal{C} \mid x \leq a, \text{ for some } a \in \mathcal{A}\}$, the ideal and filter generated by $\mathcal{A}$. Any element in $\mathcal{C} \setminus \mathcal{A}^+$ is called *independent of* $\mathcal{A}$, and we let $\mathcal{I}(\mathcal{A})$ denote the set of all maximal independent elements for $\mathcal{A}$. Given $\mathcal{A} \subseteq \mathcal{C}$ and a subset $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ of maximal independent elements of $\mathcal{A}$, problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ calls for generating a new element $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$, or proving that there is no such element. By performing $|\mathcal{I}(\mathcal{A})| + 1$ calls to problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$, we can solve the following problem

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

Problem GEN$(\mathcal{C}, \mathcal{A})$ has several interesting applications in integer programming and data mining, see [3,4,5] and the references therein. Extensions of the two hypergraph transversal algorithms mentioned above to solve problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ were given in [3]. In this paper, we give an implementation of the first dualization algorithm in [3], which achieves efficiency in two directions:

- Re-use of the recursion tree: dualization-based techniques generate all maximal independent elements of a given subset $\mathcal{A} \subseteq \mathcal{C}$ by usually performing $|\mathcal{I}(\mathcal{A})| + 1$ calls to problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$, 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 $\mathcal{I}(\mathcal{A})$, since the recursion trees required to generate many elements may be nearly identical.
- Efficient implementation at each recursion tree node: Straight forward implementation of the algorithm in [3] requires $O(n|\mathcal{A}| + n|\mathcal{B}|)$ time per recursive

call. However, this can be improved to $O(n|A|+|B|+n\log(|\mathcal{B}|))$ by maintaing a binary search tree on the elements of $\mathcal{B}$, and using randomization. Since $|\mathcal{B}|$ is usually much larger than $|\mathcal{A}|$, this gives a significant improvement.
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 $\mathcal{A}$ and $\mathcal{B}$ 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 be very effective in practically improving the running time of the algorithm.

The rest of this paper is organized as follows. In section 2 we introduce some basic terminology used throughout the paper, and briefly outline the Fredman-Khachiyan algorithm (or more precisely, its generalization to boxes). Section 3 describes the data structure used in our implementation, and Section 4 presents the algorithm. In Section 5, we show that the new version of the algorithm has, on the expected, the same quasi-polynomial bound on the running time as that of [3], and we also show how to get a slightly stronger bound on the running time. Section 6 briefly outlines our preliminary experimental findings with the new implementation for generating hypergraph transversals. Finally, we draw some conclusions in Section 7.

2 Terminology and Outline of the Algorithm

Throughout the paper, we assume that we are given an integer box $\mathcal{C}^* = \mathcal{C}_1^* \times \ldots \times \mathcal{C}_n^*$, where $\mathcal{C}_i^* = [l_i^* : u_i^*]$, and $l_i^* \leq u_i^*$, are integers, and a subset $\mathcal{A}^* \subseteq \mathcal{C}^*$ of vectors for which it is required to generate all maximal independent elements. The algorithm of [3], considered in this paper, solves problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$, 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 $\mathcal{C}$ of the original box $\mathcal{C}^*$ and two subsets $\mathcal{A} \subseteq \mathcal{A}^*$ and $\mathcal{B} \subseteq \mathcal{B}^*$ of integral vectors, where $\mathcal{B}^* \subseteq \mathcal{I}(\mathcal{A}^*)$ 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:

$$a \not\leq b, \quad \text{for all } a \in \mathcal{A}, b \in \mathcal{B}. \tag{1}$$

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

A key lemma, on which the algorithm in [3] is based, is that either (i) there is an element $x \in \mathcal{A} \cup \mathcal{B}$ with at most $1/\epsilon$ essential coordinates, where $\epsilon \stackrel{\text{def}}{=} 1/(1 + \log m)$ and $m \stackrel{\text{def}}{=} |\mathcal{A}| + |\mathcal{B}|$, or (ii) one can easily find a new maximal

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

3 The Data Structure

We use the following data structures in our implementation:

- Two arrays of vectors, A and B containing the elements of $\mathcal{A}^*$ and $\mathcal{B}^*$ respectively.
- Two (dynamic) arrays of indices, index$(\mathcal{A})$ and index$(\mathcal{B})$, containing the indices of vectors from $\mathcal{A}^*$ and $\mathcal{B}^*$ (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 $\mathcal{A}$ and $\mathcal{B}$, and also to keep track of which vectors are currently active, i.e, intersect the current box.
- A balanced binary search tree $\mathbf{T}(\mathcal{B}^*)$, built on the elements of $\mathcal{B}^*$ 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 $x \in \mathcal{C}$ belongs to $\mathcal{B}^*$ or not, takes only $O(n \log |\mathcal{B}^*|)$ time.

4 The Algorithm

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

- $\max_{\mathcal{A}}(z)$. It takes as input a vector $z \notin \mathcal{A}^+$ and returns a maximal vector z^* in $(\mathcal{C}^* \cap \{z\}^+) \setminus \mathcal{A}^+$. This can be done in $O(n|\mathcal{A}|)$ by initializing $c(a) = |\{i \in$

$[n] : a_i > z_i\}|$ for all $a \in \mathcal{A}$, and repeating, for $i = 1, \ldots, n$, the following two steps: (i) $z_i^* \leftarrow \min(u_i^*, \min\{a_i - 1 : a \in \mathcal{A},\ c(a) = 1 \text{ and } a_i > z_i\})$ (where we assume $\min(\emptyset) = \infty$); (ii) $c(a) \leftarrow c(a) - 1$ for each $a \in \mathcal{A}$ such that $z_i < a_i \leq z_i^*$.

- Exhaustive duality$(\mathcal{C}, \mathcal{A}, \mathcal{B})$. Assuming $|\mathcal{A}||\mathcal{B}| \leq 1$, check duality in $O(n(|\mathcal{A}^*| + \log|\mathcal{B}|))$ as follows: First, if $|\mathcal{A}| = |\mathcal{B}| = 1$ then find an $i \in [n]$ such that $a_i > b_i$, where $\mathcal{A} = \{a\}$ and $\mathcal{B} = \{b\}$. (Such a coordinate is guaranteed to exist by (1).) If there is a $j \neq i$ such that $b_j < u_j$ then return $\max_{\mathcal{A}^*}(u_1, \ldots, u_{i-1}, b_i, u_{i+1}, \ldots, u_n)$. 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 $(u_1, \ldots, u_{i-1}, a_i - 1, u_{i+1}, \ldots, u_n)$. Otherwise return *FALSE* (meaning that $\mathcal{A}$ and $\mathcal{B}$ are dual in $\mathcal{C}$). Second, if $|\mathcal{A}| = 0$ then let $z = \max_{\mathcal{A}^*}(u)$, and return either *FALSE* or z depending on whether $z \in \mathcal{B}^*$ or not (this check can be done in $O(n \log|\mathcal{B}^*|)$ using the search tree $\mathbf{T}(\mathcal{B}^*)$). Finally, if $|\mathcal{B}| = 0$ then return either *FALSE* or $z = \max_{\mathcal{A}^*}(l)$ depending on whether $l \in \mathcal{A}^+$ or not (this check requires $O(n|\mathcal{A}|)$ time).
- Random solution$(\mathcal{C}, \mathcal{A}^*, \mathcal{B})$. Repeat the following for $k = 1, \ldots, t_1$ times, where t_1 is a constant (say 10): Find a random point $z^k \in \mathcal{C}$, by picking each coordinate z_i^k randomly from $\{l_i, u_i\}$, $i = 1, \ldots, n$. Let $(z^k)^* \leftarrow \max_{\mathcal{A}^*}(z^k)$. If $(z^k)^* \notin \mathcal{B}^*$ then return $(z^k)^*$. If $\{(z^1)^*, \ldots, (z^{t_1})^*\} \subseteq \mathcal{B}^*$ then return *FALSE*. This step takes $O(n(|\mathcal{A}^*| + \log|\mathcal{B}^*|))$ time, and is is used to check whether $\mathcal{A}^+ \cup \mathcal{B}^-$ covers a large portion of $\mathcal{C}$.
- Count estimation. For a subset $\mathcal{X} \subseteq \mathcal{A}$ (or $\mathcal{X} \subseteq \mathcal{B}$), use sampling to estimate the number $\text{Est}(\mathcal{X}, \mathcal{C})$ of elements of $\mathcal{X} \subseteq \mathcal{A}$ (or $\mathcal{X} \subseteq \mathcal{B}$) that are active with respect to the current box $\mathcal{C}$, i.e. the elements of the set $\mathcal{X}' \stackrel{\text{def}}{=} \{a \in \mathcal{X} \mid a^+ \cap \mathcal{C} \neq \emptyset\}$ ($\mathcal{X}' \stackrel{\text{def}}{=} \{b \in \mathcal{X} \mid b^- \cap \mathcal{C} \neq \emptyset\}$). This can be done as follows. For $t_2 = O(\log(|\mathcal{A}| + |\mathcal{B}|)/\epsilon)$, pick elements $x^1, \ldots, x^{t_2} \in \mathcal{A}$ at random, and let the random variable $Y = \frac{|\mathcal{A}|}{t_2} * |\{x^i \in \mathcal{X}' : i = 1, \ldots .t_2\}|$. Repeat this step independently for a total of $t_3 = O(\log(|\mathcal{A}| + |\mathcal{B}|))$ times to obtain t_3 estimates $Y^1, \ldots, Y^{t_3}$, and let $\text{Est}(\mathcal{X}, \mathcal{C}) = \min\{Y^1, \ldots, Y^{t_3}\}$. This step requires $O(n \log^3 m)$ time. [1]
- Cleanup($\mathcal{A}$,$\mathcal{C}$) (Cleanup($\mathcal{B}$,$\mathcal{C}$)). Set $\mathcal{A}' \leftarrow \{a \in \mathcal{A} \mid a^+ \cap \mathcal{C} \neq \emptyset\}$ (respectively, $\mathcal{B}' \leftarrow \{b \in \mathcal{B} \mid b^- \cap \mathcal{C} \neq \emptyset\}$), and return $\mathcal{A}'$ (respectively, $\mathcal{B}'$). This step takes $O(n|\mathcal{A}|)$ (respectively, $O(n|\mathcal{B}|)$).

Now, we describe the implementation of procedure GEN-DUAL$(\mathcal{A}, \mathcal{B}, \mathcal{C})$ which is called initially using $\mathcal{C} \leftarrow \mathcal{C}^*$, $\mathcal{A} \leftarrow \mathcal{A}^*$ and $\mathcal{B} \leftarrow \emptyset$. At the return of this call, $\mathcal{B}$ is extended by the elements in $\mathcal{I}(\mathcal{A}^*)$. Below we assume that $f \in (0, 1)$ is a constant.

[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.

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

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

5 Analysis of the Expected Running Time

Let $C(v)$ be the expected number of recursive calls on a subproblem GEN-DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ of volume $v \stackrel{\text{def}}{=} |\mathcal{A}||\mathcal{B}|$. Consider a particular recursive call of the algorithm and let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ 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 $x^* \in \mathcal{A}$. As mentioned before, we assume also that the factor f used in Steps 6.3 and 6.8 is 1/2. For $i = 1, \ldots, n$, let $\mathcal{B}_i \stackrel{\text{def}}{=} \{b \in \mathcal{B} \ : \ b_i < x^*_i\}$, and denote by $\overline{\mathcal{B}} = \mathcal{B} \cap \mathcal{C}^+$ and $\overline{\mathcal{B}_i} = \mathcal{B}_i \cap \mathcal{C}^+$ the

subsets of $\mathcal{B}$ and $\mathcal{B}_i$ that are active with respect to the current box $\mathcal{C}$. 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 [3].

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

$$\Pr\left[|\overline{\mathcal{B}_i}| \geq \epsilon\frac{|\overline{\mathcal{B}}|}{4}\right] \geq 1 - \frac{1}{\epsilon v}. \tag{2}$$

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

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

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

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

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 $Y_i \geq Y_k$ and thus, with probability at least $1 - 1/(\epsilon v)$, we have $|\overline{\mathcal{B}_i}|/|\overline{\mathcal{B}}| \geq \epsilon/4$. □

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

Proof. For a node N of the recursion tree, denote by $\overline{\mathcal{A}} = \overline{\mathcal{A}}(N)$, $\overline{\mathcal{B}} = \overline{\mathcal{B}}(N)$ the subsets of $\mathcal{A}$ and $\mathcal{B}$ intersecting the box specified by node N, and let $v(N) = |\overline{\mathcal{A}}(N)||\overline{\mathcal{B}}(N)|$. Now consider the node N at which the lastest maximal independent element was generated. If $s \stackrel{\text{def}}{=} \sum_{a \in \overline{\mathcal{A}}(N)} (1/2)^{|\,\mathrm{Ess}(a)|} + \sum_{b \in \overline{\mathcal{B}}(N)} (1/2)^{|\,\mathrm{Ess}(b)|} < 1/2$, then the probability that the point $z \in \mathcal{C}$, picked randomly in Steps 3 or 4.2 of the procedure, belongs to $\overline{\mathcal{A}}(N)^+ \cup \overline{\mathcal{B}}(N)^-$ is at most $\sigma_1 \stackrel{\text{def}}{=} (1/2)^{t_1}$. Thus, in this case, with probability at least $1 - \sigma_1$, we find a new maximal independent element. Assume therefore that $s \geq 1/2$, let x^* be

the element with $|\operatorname{Ess}(x^*)| \leq 1/\epsilon$ found in Step 5, and assume without loss of generality that $x^* \in \mathcal{A}$. Then, by (1), there exists a coordinate $k \in \operatorname{Ess}(x^*)$ such that $|\overline{\mathcal{B}_k}| \geq \epsilon|\overline{\mathcal{B}}|$. By Lemma 1, with probability at least $\sigma_2 \stackrel{\text{def}}{=} 1 - 1/(\epsilon v)$, we can reduce the volume of one of the subproblems, of the current problem, by a factor of at least $1 - \epsilon/4$. Thus for the expected number of recursive calls at node N, we get the following recurrence

$$C(v) \leq \frac{1}{\sigma_2}\left[1 + C(v-1) + C((1-\frac{\epsilon}{4})v)\right], \tag{3}$$

where $v = v(N)$. This recurrence gives $C(v) \leq v^{O(\log^2 v)}$. Now consider the path $N_0 = N, N_1, \ldots, N_r$ from node N to the root of the recursion tree N_r. Since a large number of new maximal independent elements may have been added at node N (and of course to all its ancestors in the tree), recurrence (3) may no longer hold at nodes $N_1, \ldots, N_r$. However, since we count the number of recursive calls from the time of the last generation that happened at node N, each node N_i, that has N_{i-1} as a right child in the tree, does not contribute to this number. Furthermore, the number of recursive calls resulting from the right child of each node N_i, that has N_{i-1} as a left child, is at most $C(v(N_r))$. Since the number of such nodes does not exceed the depth of the tree, which is at most nm, the expected total number of recursive calls is at most $nmC(v(N_r))$ and the lemma follows. □

We show further that, if $|\mathcal{B}| >> |\mathcal{A}|$, 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$(\mathcal{A}, \mathcal{B}, \mathcal{C})$, is $nm^{o(log^2 m)}$.

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

$$2^c\left(a^{c/\log b} - 1\right) = 1. \tag{4}$$

Proof. Let $r = \min\{|\operatorname{Ess}(y)| \ : \ y \in \mathcal{A} \cup \mathcal{B}\}$, $p = \left(1 + \left(\frac{\beta}{\alpha}\right)^{\frac{1}{r-1}}\right)^{-1}$, and let $z \in \mathcal{C}$ be a random element obtained by picking each coordinate independently with $\Pr[z_i = l_i] = p$ and $\Pr[z_i = u_i] = 1 - p$. Then the probability that $z \in \mathcal{A}^+ \cup \mathcal{B}^-$ is at most $\sum_{a\in\mathcal{A}}(1-p)^{|\operatorname{Ess}(a)|} + \sum_{b\in\mathcal{B}} p^{|\operatorname{Ess}(b)|} \leq \alpha(1-p)^r + \beta p^r = \beta p^{r-1}$. Since $\mathcal{A}$ and $\mathcal{B}$ are dual in $\mathcal{C}$, it follows that $\beta p^{r-1} \geq 1$, and thus

$$r - 1 \leq \frac{\log \beta}{\log(1 + (\beta/\alpha)^{\frac{1}{r-1}})}. \tag{5}$$

The maximum value that r can achieve is when both sides of (5) 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

$$r' = 1 + \frac{\log(\beta/\alpha)}{c(\alpha, \beta/\alpha)}, \tag{6}$$

where $c(\cdot,\cdot)$ is as defined in (4). The case for $\alpha > \beta$ is similar and the lemma follows from (1) and Lemma 2. □

Note that, if β is much larger than α, then the root r' in (6) 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($\mathcal{A},\mathcal{B},\mathcal{C}$), from the time of last output till termination, is $nm^{o(log^2 m)}$. In fact, one can use Lemma 2 together with the method of conditional expectations to obtain an incremental deterministic algorithm for solving problem GEN($\mathcal{C},\mathcal{A}$), whose delay between any two successive outputs is of the order given by Lemma 2.

6 Experimental Results

We performed a number of experiments to evaluate our implementation. Five 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 $M(n)^d$, 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 $\{\{i,j\} : 1 \le i < j \le n,\ j \text{ is even}\}$ (i.e., for $j = 2,4,\ldots,n$, there is an edge between i and j for all $i < j$). The reason 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 $\mathcal{H}$ on n vertices obtained from the threshold graph and its dual $TH(n-2)$, $TH(n-2)^d \subseteq 2^{[n-2]}$ as follows:

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

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

$\mathcal{H}_1, \ldots, \mathcal{H}_k$. Finally, we define the hypergraph $SDFP(k)$ to be the hypergraph of $1 + 7k + 7^k$ hyperedges on n vertices, obtained by self-dualizing $\mathcal{H}$ as we did for threshold graphs.

Table 1. Performance of the algorithm for different classes of hypergraphs. Numbers below parameters indicate the total CPU time, in seconds, taken to generate all transversals.

$R(n, \alpha, d)$ $2 \le d \le n-1$	$n = 30$			$n = 50$			$n = 60$		
	$\alpha = 275$	$\alpha = 213$	$\alpha = 114$	$\alpha = 507$	$\alpha = 441$	$\alpha = 342$	$\alpha = 731$	$\alpha = 594$	$\alpha = 520$
	0.1	0.3	3.1	43.3	165.6	1746.8	322.2	2220.4	13329.5
$M(n)$	$n = 20$	$n = 24$	$n = 28$	$n = 30$	$n = 32$	$n = 34$	$n = 36$	$n = 38$	$n = 40$
	0.3	1.4	7.1	17.8	33.9	80.9	177.5	418.2	813.1
$MD(n)$	$n = 20$	$n = 24$	$n = 28$	$n = 30$	$n = 32$	$n = 34$	$n = 36$	$n = 38$	$n = 40$
	0.15	1.3	13.3	42.2	132.7	421.0	1330.3	4377.3	14010.5
$TH(n)$	$n = 40$	$n = 60$	$n = 80$	$n = 100$	$n = 120$	$n = 140$	$n = 160$	$n = 180$	$n = 200$
	0.4	1.9	6.0	18.4	40.2	78.2	142.2	232.5	365.0
$SDTH(n)$	$n = 42$	$n = 62$	$n = 82$	$n = 102$	$n = 122$	$n = 142$	$n = 162$	$n = 182$	$n = 202$
	0.9	5.9	23.2	104.0	388.3	1164.2	2634.0	4820.6	8720.0
$SDFP(n)$	$k = 16$		$n = 23$		$n = 30$		$n = 37$		
	0.1		4.8		198.1		11885.1		

The experiments were performed on a *Pentium 4* processor with 2.2 GHz of speed and 512M bytes of memory. Table 1 summarizes our results for several instances of the different classes of hypergraphs listed above. In the table, we show the total CPU time, in seconds, required to generate all transversals for the specified hypergraphs, with the specified parameters. For random hypergraphs, the time reported is the average over 30 experiments. The average sizes of the transversal hypergraphs, corresponding to the random hypergraphs in Table 1 are (from left to right): 150, 450, $5.7 * 10^3$, $1.7 * 10^4$, $6.4 * 10^4$, $4.7 * 10^5$, $7.5 * 10^4$, $4.7 * 10^5$, and $1.7 * 10^6$, respectively. The output sizes for the other classes of hypergraphs can be computed using the formulas given above. For instance, for $SDTH(n)$, with $n = 202$ vertices, the number of hyperedges is $\alpha = 10102$. For random hypergraphs, we only show results for $n \le 60$ 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, Figure 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 Figure 2, we show that the randomized implementation of the algorithm offers substantial improvement over the deterministic one. Figures 3 and 4, respectively, 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.

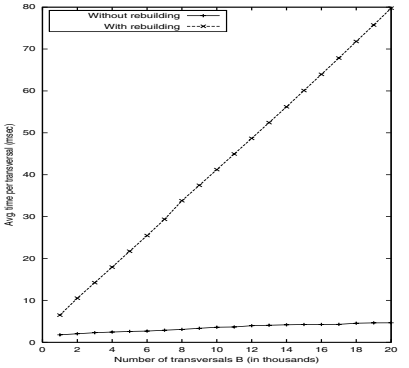

Fig. 1. Effect of rebuilding the recursion tree. Each plot shows the average CPU time (in milli-seconds) per generated transversal versus the number of transversals, for hypergraphs of type $R(30, 100, 5)$.

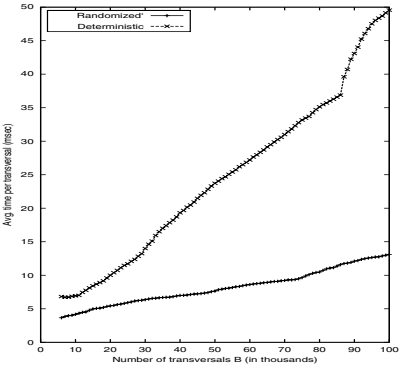

Fig. 2. Comparing deterministic versus randomized implementations. Each plot shows the average CPU time/transversal versus the number of transversals, for hypergraphs of type $R(50, 100, 10)$.

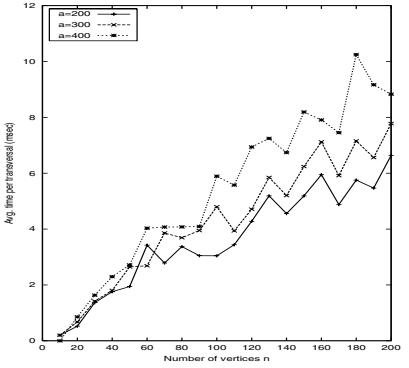

Fig. 3. Average CPU time/transversal versus the number of vertices n for random hypergraphs $R(n, a, d)$, where $d = n/4$, and $a = 200, 300, 400$.

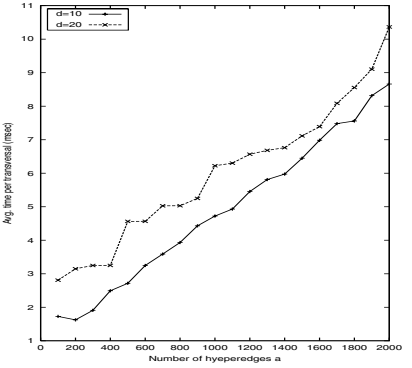

Fig. 4. Average CPU time/transversal versus the number of hyperedges a for random hypergraphs $R(50, a, d)$, for $d = 10, 20$.

7 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 [13] in which a heuristic for solving this problem was described and experimentally evaluated. However, the results in [13] show the performance for relatively small instances which are easy cases for our implemen-

tation. 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 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.

Acknowledgements. We thank the referees for the helpful remarks.

References

1. M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge Univ. Press, 1992.
2. R. Agrawal, T. Imielinski and A. Swami, Mining associations between sets of items in massive databases, *Proc. 1993 ACM-SIGMOD Int. Conf.*, pp. 207–216.
3. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K.Makino, Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities, *SIAM Journal on Computing*, **31** (5) (2002) pp. 1624–1643.
4. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, Generating Dual-Bounded Hypergraphs, *Optimization Methods and Software*, (OMS) **17** (5), Part I (2002), pp. 749–781.
5. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K.Makino, An intersection inequality for discrete distributions and related generation problems, to appear in *ICALP 2003*.
6. E. Boros, V. Gurvich, L. Khachiyan and K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets, in *19th Int. Symp. on Theoretical Aspects of Computer Science, (STACS)*, March 2002, LNCS 2285, pp. 133–141.
7. C. J. Colbourn, *The combinatorics of network reliability*, Oxford Univ. Press, 1987.
8. T. Eiter and G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24 (1995) pp. 1278–1304.
9. M. L. Fredman and L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, *Journal of Algorithms*, 21 (1996) pp. 618–628.
10. D. Gunopulos, R. Khardon, H. Mannila and H. Toivonen, Data mining, hypergraph transversals and machine learning, in *Proc. 16th ACM-PODS Conf.*, (1997) pp. 209–216.
11. V. Gurvich, To theory of multistep games, *USSR Comput. Math. and Math Phys.* **13-6** (1973), pp. 1485–1500.
12. V. Gurvich, Nash-solvability of games in pure strategies, *USSR Comput. Math. and Math. Phys.*, **15** (1975), pp. 357–371.
13. D. J. Kavvadias and E. C. Stavropoulos, Evaluation of an algorithm for the transversal hypergraph problem, in *Proc. 3rd Workshop on Algorithm Engineering (WAE'99)*, LNCS 1668, pp. 72–84, 1999.
14. R. C. Read, Every one a winner, or how to avoid isomorphism when cataloging combinatorial configurations, *Annals of Disc. Math.* 2 (1978) pp. 107–120.