Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 9, no. 2, pp. 239–264 (2005)

An Efficient Algorithm for the Transversal Hypergraph Generation

Dimitris J. Kavvadias

University of Patras
Department of Mathematics
GR-265 04 Rio, Patras, Greece
djk@math.upatras.gr

Elias C. Stavropoulos

University of Patras
Computer Engineering & Informatics Department
GR-265 04 Rio, Patras, Greece
http://lca.ceid.upatras.gr/∼estavrop
estavrop@ceid.upatras.gr

Abstract

The Transversal Hypergraph Generation is the problem of generating, given a hypergraph, the set of its minimal transversals, i.e., the hypergraph whose hyperedges are the minimal hitting sets of the given one. The purpose of this paper is to present an efficient and practical algorithm for solving this problem. We show that the proposed algorithm operates in a way that rules out regeneration and, thus, its memory requirements are polynomially bounded to the size of the input hypergraph. Although no time bound for the algorithm is given, experimental evaluation and comparison with other approaches have shown that it behaves well in practice and it can successfully handle large problem instances.

Article Type	Communicated by	Submitted	Revised
regular paper	Alan Gibbons	March 2003	August 2005

1 Introduction

Hypergraph theory [3] is an important area of discrete mathematics with a large number of applications in both theoretical and applied Computer Science. A *hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a finite collection $\mathcal{E}$ of sets over a finite set $\mathcal{V}$. The elements of $\mathcal{V}$ are called *nodes* while the elements of $\mathcal{E}$ are called *hyperedges*. A *transversal* (or *hitting set*) of $\mathcal{H}$ is a set $\mathcal{T} \subseteq \mathcal{V}$ that has a non-empty intersection with every hyperedge of $\mathcal{H}$. A transversal $\mathcal{T}$ is *minimal* if no proper subset of $\mathcal{T}$ is a hitting set of $\mathcal{H}$. The collection of all minimal transversals of $\mathcal{H}$, denoted by $Tr(\mathcal{H})$, is called the *transversal hypergraph* of $\mathcal{H}$.

The Transversal Hypergraph Generation is the problem of generating the transversal hypergraph $Tr(\mathcal{H})$ of a given hypergraph $\mathcal{H}$. Its decisional variant, Transversal Hypergraph, is the problem of deciding whether, given two hypergraphs $\mathcal{H}$ and $\mathcal{G}$ defined on the same set of nodes, $\mathcal{G} = Tr(\mathcal{H})$ holds.

Transversal Hypergraph Generation is one of the most important problems on hypergraphs with many practical applications in various areas of Computer Science, especially in Logic and Artificial Intelligence. For example, there are certain problems in propositional circumscription [8], in model-based diagnosis [10], in model-preference default reasoning [26, 37, 38, 39], and in machine learning [7, 12, 22, 23] that are reduced to solving a Transversal Hypergraph Generation problem. For an exposition of applications of the Transversal Hypergraph Generation see [14, 15, 18]. An interesting relation between the Transversal Hypergraph Generation and the field of knowledge discovery in databases was pointed out in [22, 23]. Recent applications of the problem are channel assignment in cellular mobile communication systems [36] and computing homology groups of finite simplicial complexes in Topology [13].

The main reason for the large applicability of the Transversal Hypergraph Generation problem is that finding minimal or maximal (with respect to some property) structures or solutions is a common and essential task in many areas. The notion of the transversal is a nice way of modelling these extremal structures. Even more, there are many natural problems that are just a disguised form of the Transversal Hypergraph Generation. Such an example is the generation of all prime implicants of the *dual form* of a monotone Boolean expression in DNF [18, 24]. Another important problem is the generation of all *maximal models* of Boolean expression in CNF, having all its variables negated [28]. The above problems are polynomially equivalent to Transversal Hypergraph Generation while the generation of the maximal models of any Boolean expression in CNF is at least as hard as the Transversal Hypergraph Generation.

It is easy to see that a hypergraph $\mathcal{H}$ may have exponentially many (with respect to its size) minimal transversals. Thus, an algorithm that solves a generation problem with large output, like the Transversal Hypergraph Generation, may require exponentially many steps to produce the whole output. There is a surge of interest in defining suitable complexity measures for the efficiency of a generation algorithm. *Total-polynomiality* or *output-polynomiality* is a measure that takes into account not only the size of the input but *the size of the output*, too. Stronger requirements for the efficiency of a generation algorithm take into account the size of the input and the size of the output so far (*incrementally output-polynomial algorithm*) or the delay time between consecutive outputs (*polynomial delay algorithm*). For further discussions on performance criteria for problems with large output see [20, 25, 27, 34].

Complexity questions related to the generation of minimal transversals have been widely discussed in the literature (see, for example, [9, 4, 14, 17, 18, 28, 30]). However, the exact complexity of the Transversal Hypergraph Generation problem is still open. Its complexity strongly depends on the complexity of its decision version Transversal Hypergraph since there would exist an output-polynomial time algorithm for solving the Transversal Hypergraph Generation problem if and only if the Transversal Hypergraph problem was polynomial time solvable [4]. The Transversal Hypergraph problem is in its generality in co-NP, while several polynomial time cases also exist (see [14, 15] for more details and references on them). Although there are several algorithms that involve, in some manner, the computation of minimal transversals (see, for example, [2, 32, 33, 35, 36]), no output-polynomial time algorithm is known. In 1996, Fredman and Khachiyan [18] presented an algorithm for solving the decision version in subexponential time $n^{o(\log n)}$, where n is the combined size of the input, i.e., the number of hyperedges of the two hypergraphs. This algorithm can be used as an oracle for solving the Transversal Hypergraph Generation in *incremental output-subexponential time* [24], the best provable upper time bound yet.

The result of Fredman and Khachiyan in [18] implies that the Transversal Hypergraph problem is not co-NP–hard, unless any co-NP–complete problem can be solved in subexponential time, and gives evidence that it lies in an intermediate class between P and co-NP. It was recently shown in [16, 17] and, independently, in [30] that the complement of the Transversal Hypergraph problem can be solved by a nondeterministic algorithm that makes polynomially many deterministic steps plus $O(\log^2 n)$ nondeterministic ones. This result places the Transversal Hypergraph problem in the class co-NP[$\log^2 n$], the subclass of co-NP where only the first $O(\log^2 n)$ steps are nondeterministic (see [11, 21, 31] for more on *limited nondeterminism*). It also makes straightforward the subexponential running time of the algorithm presented in [18].

In this paper we present an algorithm for solving the Transversal Hypergraph Generation problem. The proposed algorithm computes all minimal transversals of the input hypergraph correctly and efficiently and, hence, it is suitable for solving problems that can be modelled as a Transversal Hypergraph Generation. The aim of this paper is to give a detailed description of the algorithm, to prove its correctness and to study its time and space complexity. An early version of the proposed algorithm was firstly presented and experimentally evaluated in [29]. At that time no implementation of any algorithm for solving the problem was known to us. After the first submission of this work, two implementations were published: first, the implementation of the algorithm of Fredman and Khachiyan [18] by Boros et al. in [6], and second, the implementation of an algorithm proposed by Bailey et al. in [1]. Thus, we have revised the code and made several improvements in the auxiliary data structures even though the main algorithm is unchanged. Then, a new experimental study was made to evaluate and compare the proposed algorithm with the above implementations. The computational results are incorporated here.

Our algorithm is based on the brute force scheme given by Berge in [3] (we describe this scheme in the following section). This simple method needs exponential many steps to produce the whole output, it generates the first minimal transversal near the end of the procedure and its high memory requirements make it suitable only for small problem instances. On the contrary, experimental evaluation on a number of test cases have shown that the proposed algorithm is more effective and efficient even for large problem instances. Compared with the scheme of Berge, it generates all minimal transversals quite fast, while it presents a notable uniformity in the rate of the output (unfortunately, no time bound for the delay between consecutive outputs is currently proven). Although no time bound is given, we prove that its space complexity is polynomially bounded by the size of the input hypergraph (where, as usual, the size of the output does not count in the total space requirements of the algorithm). To our knowledge, this is the first algorithm that achieves space bound that is a polynomial to the size of the input hypergraph. This happens because it operates in a *generate-and-forget* fashion i.e., no previous minimal transversal is required for the generation of the next ones. In contrast, other algorithms require that all generated minimal transversals must be stored. This means that in case of a large output, memory requirements could be devastating. In addition, absolute time delays are very small, allowing the successful handling of large problem instances with large output.

The rest of this paper is organized as follows: In the next section we give some formal definitions and notations, along with the necessary properties on hypergraphs. The simple scheme of Berge is also described there. In Section 3 we give a modified version of the algorithm of Berge by utilizing the notion of the *generalized node*. A further modification is given next that computes minimal transversals in a depth-first search manner. To avoid regenerations, we define in Section 4 the new concept of the *appropriate node* and we next describe the final improved version of the algorithm and prove its correctness and its space complexity. Some implementation issues of our code are discussed in Section 5 and experimental evaluation and comparison results concerning the algorithms previously mentioned are also given there. Finally, in the last section conclusions and directions for future work are given.

2 Preliminaries

In this section we give some formal definitions and the necessary properties on hypergraphs. For more theoretical issues the reader is referred to [3, 5].

Definition 1 *A* hypergraph $\mathcal{H}$ *is an ordered pair* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$*, where* $\mathcal{V} =$

$\{v_1, \ldots, v_n\}$ is a finite set of elements and $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ is a family of subsets of $\mathcal{V}$ such that

1. *$\mathcal{E}_i \neq \emptyset$ $(i = 1, \ldots, m)$ and*
2. *$\cup_{i=1}^{m} \mathcal{E}_i = \mathcal{V}$.*

The elements of $\mathcal{V}$ are called nodes *while the elements of $\mathcal{E}$ are called* hyperedges *of the hypergraph $\mathcal{H}$.*

A hypergraph can be seen as a generalization of a graph where the restriction of an edge having only two nodes does not hold. For convenience, we shall identify a hypergraph $\mathcal{H}$ on a node set $\mathcal{V}$ with its set of edges $\mathcal{E}$, if there is no danger of ambiguity.

Definition 2 *Let $\mathcal{H} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m\}$ be a hypergraph on $\mathcal{V}$. The* partial hypergraph *$\mathcal{H}_i$ of $\mathcal{H}$ $(i = 1, \ldots, m)$ is the hypergraph on $\mathcal{V}$ that contains the first i hyperedges of $\mathcal{H}$, i.e., $\mathcal{H}_i = \{\mathcal{E}_1, \ldots, \mathcal{E}_i\}$.*

Definition 3 *A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is* simple *if for every pair $\mathcal{E}_i, \mathcal{E}_j \in \mathcal{E}$, $\mathcal{E}_j \subseteq \mathcal{E}_i \Rightarrow j = i$.*

Definition 4 *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. Then, $Min(\mathcal{H})$ is the set of minimal hyperedges of $\mathcal{H}$ with respect to set inclusion, i.e., $Min(\mathcal{H}) = \{\mathcal{E}_i \in \mathcal{E} \mid (\forall \mathcal{E}_j \in \mathcal{E}, i \neq j, \mathcal{E}_j \subseteq \mathcal{E}_i) : \mathcal{E}_j = \mathcal{E}_i\}$, and $Max(\mathcal{H})$ is the set of maximal hyperedges of $\mathcal{H}$ with respect to set inclusion, i.e., $Max(\mathcal{H}) = \{\mathcal{E}_i \in \mathcal{E} \mid (\forall \mathcal{E}_j \in \mathcal{E}, j = 1, \ldots, m, i \neq j, \mathcal{E}_j \supseteq \mathcal{E}_i) : \mathcal{E}_j = \mathcal{E}_i\}$.*

Simple hypergraphs are also known as *Sperner families* [3, 5]. It is easy to see that for any hypergraph $\mathcal{H}$ on $\mathcal{V}$, $Min(\mathcal{H})$ and $Max(\mathcal{H})$ are simple hypergraphs and can be computed in time that is a polynomial in the number of hyperedges of $\mathcal{H}$. Moreover, every partial hypergraph of a simple hypergraph is simple, too.

Definition 5 *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A set $\mathcal{T} \subseteq \mathcal{V}$ is called a* transversal *(or,* hitting set*) of $\mathcal{H}$ if it intersects all its hyperedges, i.e., $\mathcal{T} \cap \mathcal{E}_i \neq \emptyset, \forall \mathcal{E}_i \in \mathcal{E}$. A transversal $\mathcal{T}$ is called* minimal *if no proper subset $\mathcal{T}'$ of $\mathcal{T}$ is a transversal of $\mathcal{H}$.*

In graphs, a transversal is usually called a node cover. If $\mathcal{T}$ is a transversal of $\mathcal{H}$ on $\mathcal{V}$, then the complementary set $\mathcal{I} = \mathcal{V} \setminus \mathcal{T}$ of $\mathcal{T}$ forms an *independent set* of $\mathcal{H}$, i.e., a set that does not contain any hyperedge of $\mathcal{H}$. (This reduces to our standard notion of independent set in graphs if we specialize to the case where all hyperedges contain precisely two nodes.) A minimal transversal can be identified in polynomial time by removing, starting from $\mathcal{V}$, one by one the nodes of $\mathcal{V}$ and checking after each removal whether the remaining set is a hitting set. However, finding a transversal with minimum cardinality is NP-hard, which is a consequence of the NP-complete MINIMUM NODE COVER problem [19].

Definition 6 *The* transversal hypergraph $Tr(\mathcal{H})$ *of a hypergraph* $\mathcal{H}$ *is the family of all minimal transversals of* $\mathcal{H}$*.*

The next proposition follows from the definition of the transversal hypergraph.

Proposition 1 *Let* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ *be a hypergraph. Then, the transversal hypergraph* $Tr(\mathcal{H})$ *of* $\mathcal{H}$ *is a simple hypergraph and* $Tr(\mathcal{H}) = Tr(Min(\mathcal{H}))$*.*

It is easy to see that given two hypergraphs $\mathcal{H}$ and $\mathcal{G}$ defined on the same set of nodes $\mathcal{V}$, the problem of deciding whether $\mathcal{G}$ is the transversal hypergraph of $\mathcal{H}$ (Transversal Hypergraph) is in co-NP since a succinct disqualifier (a minimal transversal of $\mathcal{H}$ not contained in $\mathcal{G}$) can be guessed and verified in time that is a polynomial in the size of the input, i.e., to the combined number of the hyperedges of $\mathcal{H}$ and $\mathcal{G}$. It was recently shown in [16, 30] that the problem can be solved with limited nondeterminism and it was placed in the class co-NP$[\log^2 n]$, where n in the size of the input.

Proposition 2 ([14, Proposition 4.4]) *The* Transversal Hypergraph *problem is computationally equivalent to the subcase where the input hypergraphs are simple hypergraphs.*

Without loss of generality, we shall henceforth deal only with simple hypergraphs defined on the same set of nodes. The following propositions capture important relations between a hypergraph and its transversal hypergraph (for proofs see [3]).

Proposition 3 *Let* $\mathcal{H}$ *and* $\mathcal{G}$ *two simple hypergraphs. Then,*

$$\mathcal{G} = Tr(\mathcal{H}) \text{ if and only if } \mathcal{H} = Tr(\mathcal{G}). \tag{1}$$

Corollary 1 (Duality Property) *Let* $\mathcal{H}$ *be a simple hypergraph. Then,*

$$Tr(Tr(\mathcal{H})) = \mathcal{H}. \tag{2}$$

Corollary 2 *Let* $\mathcal{H}$ *and* $\mathcal{G}$ *two simple hypergraphs. Then,*

$$Tr(\mathcal{H}) = Tr(\mathcal{G}) \text{ if and only if } \mathcal{H} = \mathcal{G}. \tag{3}$$

We end this section by giving the definition of two useful operations:

Definition 7 *Let* $\mathcal{H} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ *and* $\mathcal{G} = \{\mathcal{E}'_1, \ldots, \mathcal{E}'_{m'}\}$ *be two hypergraphs. Then,*

$$\begin{aligned} \mathcal{H} \cup \mathcal{G} &= \{\mathcal{E}_1, \ldots, \mathcal{E}_m, \mathcal{E}'_1, \ldots, \mathcal{E}'_{m'}\}, \text{ and} && (4) \\ \mathcal{H} \vee \mathcal{G} &= \{\mathcal{E}_i \cup \mathcal{E}'_j,\ i = 1, \ldots, m,\ j = 1, \ldots, m'\}. && (5) \end{aligned}$$

The first operation is the *union* of $\mathcal{H}$ and $\mathcal{G}$, i.e, the hypergraph whose hyperedges are the hyperedges of both hypergraphs. The second one is in some sense the *Cartesian product* of them, i.e., the union of all possible pairs of hyperedges, one from the first hypergraph and one from the second one. We next state an important property that holds for simple hypergraphs (for a proof see [3]):

```
for $i = 2, \ldots, m$ do
  Find $Tr(\mathcal{H}_{i-1})$
  Compute $Tr(\mathcal{H}_i) = Min(Tr(\mathcal{H}_{i-1}) \vee \{\{v\}, v \in \mathcal{E}_i\})$
end for
Return $Tr(\mathcal{H}_m)$
```

Algorithm 1: The algorithm of Berge

Proposition 4 *Let $\mathcal{H}$ and $\mathcal{G}$ be two simple hypergraphs. Then,*

$$Tr(\mathcal{H} \cup \mathcal{G}) = Min(Tr(\mathcal{H}) \vee Tr(\mathcal{G})). \tag{6}$$

Based on Proposition 4, there is a simple scheme attributed to Berge [3] for generating all minimal transversals of a hypergraph $\mathcal{H} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ on $\mathcal{V}$. Let $\mathcal{H}_i = \{\mathcal{E}_1, \ldots, \mathcal{E}_i\}, i = 1, \ldots, m$ be the partial hypergraph of $\mathcal{H}$ on $\mathcal{V}$. It holds that $\mathcal{H}_i = \mathcal{H}_{i-1} \cup \{\mathcal{E}_i\}$, for all $i = 2, \ldots, m$, while $\mathcal{H}_1 = \{\mathcal{E}_1\}$ and $\mathcal{H}_m = \mathcal{H}$. Thus, $Tr(\mathcal{H}_i) = Tr(\mathcal{H}_{i-1} \cup \{\mathcal{E}_i\})$ and, according to Equation (6),

$$\begin{aligned} Tr(\mathcal{H}_i) &= Min(Tr(\mathcal{H}_{i-1}) \vee Tr(\{\mathcal{E}_i\})) \\ &= Min(Tr(\mathcal{H}_{i-1}) \vee \{\{v\}, v \in \mathcal{E}_i\}) \end{aligned} \tag{7}$$

The algorithm of Berge (see Algorithm 1) is based on Equation (7) and computes all minimal transversals of the input hypergraph $\mathcal{H}$ recursively, in two steps: First, it computes the minimal transversals of the partial hypergraph $\mathcal{H}_{i-1}$ and then it calculates the Cartesian product of the set $Tr(\mathcal{H}_{i-1})$ by the i-th hyperedge $\mathcal{E}_i$ of $\mathcal{H}$ and removes all elements that are not minimal. Thus, one can compute $Tr(\mathcal{H})$ by starting from the minimal transversals of $\mathcal{E}_1$ (note that the minimal transversals of a hypergraph with a single hyperedge are exactly its nodes) and adding one-by-one the rest of the hyperedges, computing at each step the set of minimal transversals of the new partial hypergraph. The procedure terminates after the addition of the last hyperedge $\mathcal{E}_m$. Algorithm 1 then outputs the transversal hypergraph $Tr(\mathcal{H})$ of the input hypergraph $\mathcal{H}$.

Theorem 1 *Algorithm 1 correctly generates all minimal transversals of any simple hypergraph $\mathcal{H}$.*

Proof: Follows directly from Proposition 4. □

The algorithm of Berge is the most simple and direct scheme for computing the minimal transversals of a hypergraph. However, there are several drawbacks that make it inefficient and unsuitable for large problem instances. First of all, notice that all, possibly exponentially many, intermediate transversals of the partial hypergraphs $\mathcal{H}_i$ ($i = 1, \ldots, m-1$) must be computed (the Cartesian product of the set $Tr(\mathcal{H}_{i-1})$ by the hyperedge $\mathcal{E}_i$) and only the minimal of them must be kept. This means than the total running time of the algorithm may be exponential in both the size of the input and the output. No less important are the memory requirements that also emerge from the above. All these intermediate minimal transversals have to be stored and kept until used for the computation of the new transversal set. Since the number of these intermediate minimal transversals can be exponential, the memory requirements of the algorithm can become devastating. And last but not least, since the computation of the first transversal of the input hypergraph $\mathcal{H}$ is accomplished after all minimal transversals of the partial hypergraph $\mathcal{H}_{m-1}$ have been computed, the first *final* minimal transversal is output after exponential delay time. This is the most severe drawback of the algorithm of Berge in view of the complexity measures for our problem.

3 Generalized Nodes and Depth-First Transversal Computation

In this section we describe a number of modifications and improvements to the simple scheme of Berge that make the final algorithm quite efficient, practical, and suitable for large instances.

3.1 Generalized Nodes

Our first aim is to reduce the large number of intermediate partial transversals produced by Algorithm 1. This would improve the total running time of the algorithm and reduce its storage requirements. To do this, we define the notion of the *generalized node*:

Definition 8 *Let $\mathcal{H}$ be a hypergraph on $\mathcal{V}$. The set $\mathcal{X} \subseteq \mathcal{V}$ is a* generalized node *of $\mathcal{H}$ if all nodes in $\mathcal{X}$ belong in exactly the same hyperedges of $\mathcal{H}$.*

Obviously, the cardinality of a generalized node may vary from 1 to $|\mathcal{V}|$. If $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_k$ are all the generalized nodes of $\mathcal{H}$, then $\mathcal{V} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \ldots \cup \mathcal{X}_k$, while $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$, for all $i \neq j, i, j = 1, \ldots, k$.

Definition 9 *Let $\mathcal{H} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ be a hypergraph on $\mathcal{V}$ and $\mathcal{X} \subseteq \mathcal{V}$ be a generalized node of $\mathcal{H}$. Then, the* generalized hypergraph of $\mathcal{H}$ with respect to *$\mathcal{X}$ is the hypergraph $\mathcal{H}^g_{\mathcal{X}} = \{\mathcal{E}^g_1, \ldots, \mathcal{E}^g_m\}$ on $\mathcal{V}^g_{\mathcal{X}} = \{\{\mathcal{V} \setminus \mathcal{X}\} \cup \{v_{\mathcal{X}}\}\}$, where $v_{\mathcal{X}}$ is an auxiliary node not in $\mathcal{V}$ and $\mathcal{E}^g_i$ $(1 \leq i \leq m)$ follows from $\mathcal{E}_i$ by substituting (if it appears) the set $\mathcal{X}$ by the node $v_{\mathcal{X}}$.*

The above definition can be generalized for more than one generalized nodes:

Definition 10 *If $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_k, \mathcal{X}_i \subseteq \mathcal{V}, i = 1, \ldots, k$, are the generalized nodes of the hypergraph $\mathcal{H} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ on $\mathcal{V}$, then the* generalized hypergraph of *$\mathcal{H}$ is the hypergraph $\mathcal{H}^g = \{\mathcal{E}^g_1, \ldots, \mathcal{E}^g_m\}$ on $\mathcal{V}^g = \{v_{\mathcal{X}_1}, v_{\mathcal{X}_2}, \ldots, v_{\mathcal{X}_k}\}$, where $v_{\mathcal{X}_1}, v_{\mathcal{X}_2}, \ldots, v_{\mathcal{X}_k}$ are auxiliary nodes not in $\mathcal{V}$ and $\mathcal{E}^g_i$ $(1 \leq i \leq m)$ follows from $\mathcal{E}_i$ by substituting (if they appear) the sets $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_k$ by the nodes $v_{\mathcal{X}_1}, v_{\mathcal{X}_2}, \ldots, v_{\mathcal{X}_k}$, respectively.*

Assume that the hypergraph $\mathcal{H}$ has a generalized node $\mathcal{X}$ with cardinality $|\mathcal{X}| \geq 2$. Let $\mathcal{H}^g_{\mathcal{X}}$ be the generalized hypergraph of $\mathcal{H}$ with respect to $\mathcal{X}$ and let $Tr(\mathcal{H}^g_{\mathcal{X}})$ be the transversal hypergraph of $\mathcal{H}^g_{\mathcal{X}}$. The importance of the concept of the generalized node follows from the observation that

$$\begin{aligned} Tr(\mathcal{H}) \quad = \quad & \{\mathcal{T}^g \in Tr(\mathcal{H}^g_{\mathcal{X}}) \,|\, v_{\mathcal{X}} \notin \mathcal{T}^g\} \cup \\ & \{(\mathcal{T}^g \setminus v_{\mathcal{X}}) \vee \mathcal{X}, \forall \mathcal{T}^g \in Tr(\mathcal{H}^g_{\mathcal{X}}) \,|\, v_{\mathcal{X}} \in \mathcal{T}^g\}. \end{aligned} \tag{8}$$

In other words, the minimal transversals of $\mathcal{H}$ follow by taking one by one the minimal transversals of $\mathcal{H}^g\mathcal{X}$ that include the node $v_{\mathcal{X}}$ and replacing $v_{\mathcal{X}}$ by each (simple) node in $\mathcal{X}$, in turn. Obviously, the number of minimal transversals of $\mathcal{H}$ produced from a single minimal transversal $\mathcal{T}^g$ of $\mathcal{H}^g_{\mathcal{X}}$ is exactly $|\mathcal{X}|$. The minimal transversals of $\mathcal{H}^g\mathcal{X}$ that do not include $v_{\mathcal{X}}$ remain as they are, since they hit $\mathcal{H}$. This procedure can be generalized to any number of generalized nodes.

Lemma 1 *Let $\mathcal{H}$ be a hypergraph on $\mathcal{V}$ and $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_k, \mathcal{X}_i \subseteq \mathcal{V}, i = 1, \ldots, k$, be its generalized nodes. Let also $\mathcal{T}^g = \{\mathcal{X}_{i_1}, \ldots, \mathcal{X}_{i_l}\}, 1 \leq i_1, \ldots, i_l \leq k$, be a minimal transversal of the generalized hypergraph $\mathcal{H}^g$ of $\mathcal{H}$. Then,*

1. *every l-tuple of the Cartesian product $\mathcal{X}_{i_1} \vee \mathcal{X}_{i_2} \vee \cdots \vee \mathcal{X}_{i_l}$ is a minimal transversal of $\mathcal{H}$ and*

2. *no other minimal transversal of $\mathcal{H}$ exists.*

Proof: Let $\mathcal{T} = \{v_{i_1}, \ldots, v_{i_l}\}$ be an l-tuple of the Cartesian product $\mathcal{X}_{i_1} \vee \mathcal{X}_{i_2} \vee \cdots \vee \mathcal{X}_{i_l}$ such that $v_{i_j} \in \mathcal{X}_{i_j}, j = 1, \ldots, l$. Every simple node v_{i_j} is actually a unique representative of $\mathcal{X}_{i_j}$ in $\mathcal{T}$. Since $\mathcal{T}^g$ is a transversal of $\mathcal{H}^g$ and all nodes of every generalized node of $\mathcal{H}$ belong to exactly the same hyperedges of $\mathcal{H}$, then $\mathcal{T}$ is a transversal of $\mathcal{H}$. Moreover, the removal of a simple node of $\mathcal{T}$ would result in a set that does not hit at least one hyperedge of $\mathcal{H}$ since every generalized node is represented in $\mathcal{T}$ by exactly one simple node. Hence, $\mathcal{T}$ is a minimal transversal of $\mathcal{H}$.

To prove the second statement, see that if $\mathcal{T}$ is a minimal transversal of $\mathcal{H}$, then $\mathcal{T}$ has at least one common node with every hyperedge of $\mathcal{H}$. Every node of $\mathcal{T}$ corresponds to exactly one generalized node. If $\mathcal{T}^g$ is the collection of all these generalized nodes, then $\mathcal{T}^g$ is a transversal of $\mathcal{H}^g$ since it intersects every hyperedge $\mathcal{E}^g_i$ of it. Moreover, $\mathcal{T}^g$ is minimal (a proper subset $\mathcal{T}'^g$ of $\mathcal{T}^g$ that intersects every hyperedge of $\mathcal{H}^g$ would result, by taking the Cartesian product of its nodes, in a set $\mathcal{T}'$ that is contained in $\mathcal{T}$ and intersects every hyperedge of $\mathcal{H}$, a contradiction). □

Example 1 *Assume that a hypergraph $\mathcal{H}$ has two hyperedges with 100 nodes each: $\mathcal{E}_1 = \{1, \ldots, 100\}$ and $\mathcal{E}_2 = \{51, \ldots, 150\}$. The partial hypergraph $\mathcal{H}_2 = \{\mathcal{E}_1, \mathcal{E}_2\}$ has 2550 minimal transversals (2500 with two nodes and 50 with one node) which must be kept for the subsequent stage if we use the straightforward scheme. For $\mathcal{H}_2$, three generalized nodes are defined: $\mathcal{X}_1 = \{1, \ldots, 50\}$, $\mathcal{X}_2 =$*

$\{51, \ldots, 100\}$*, and* $\mathcal{X}_3 = \{101, \ldots, 150\}$*. Using the generalized node approach, we have only* 2 *minimal transversals to store, namely the set* $\{\{51, \ldots, 100\}\}$ *and the set* $\{\{1, \ldots, 50\}, \{101, \ldots, 150\}\}$*. All minimal transversals of* $\mathcal{H}_2$ *may occur from these, as Lemma 1 suggests.*

According to Lemma 1, every minimal transversal $\mathcal{T}$ of $\mathcal{H}$ is an *offspring* of some minimal transversal $\mathcal{T}^g$ of $\mathcal{H}^g$. Thus, the generation of $Tr(\mathcal{H})$ is now reduced to the generation of $Tr(\mathcal{H}^g)$.

3.2 The Modified Algorithm of Berge

We shall now describe a modification of Algorithm 1 that exploits the concept of the generalized node explained above.

Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_{k_i}$ be the generalized nodes of the partial hypergraph $\mathcal{H}_i = \{\mathcal{E}_1, \ldots, \mathcal{E}_i\}$ of $\mathcal{H}$, $k_i \geq 1$. Assume that we have already defined the generalized nodes of $\mathcal{H}_i$ and computed $Tr(\mathcal{H}_i^g)$. We add now the next hyperedge $\mathcal{E}_{i+1}$ to define the partial hypergraph $\mathcal{H}_{i+1} = \mathcal{H}_i \cup \{\mathcal{E}_{i+1}\}$. The addition of $\mathcal{E}_{i+1}$ imposes the new determination of all previously determined generalized nodes. There are three possible types for every generalized node $\mathcal{X}$ of $\mathcal{H}_i$:

(α) $\mathcal{X} \cap \mathcal{E}_{i+1} = \emptyset$. In this case, $\mathcal{X}$ is also a generalized node of $\mathcal{H}_{i+1}$.

(β) $\mathcal{X} \subset \mathcal{E}_{i+1}$. In this case, $\mathcal{X}$ is also a generalized node of $\mathcal{H}_{i+1}$.

(γ) $\mathcal{X} \cap \mathcal{E}_{i+1} \neq \emptyset$ and $\mathcal{X} \not\subset \mathcal{E}_{i+1}$. In this case, $\mathcal{X}$ is divided into $\mathcal{X}_1 = \mathcal{X} \setminus (\mathcal{X} \cap \mathcal{E}_{i+1})$ and $\mathcal{X}_2 = \mathcal{X} \cap \mathcal{E}_{i+1}$. Both $\mathcal{X}_1$ and $\mathcal{X}_2$ are generalized nodes of $\mathcal{H}_{i+1}$.

Notice that the determination of the new set of generalized nodes depends only on the addition of $\mathcal{E}_{i+1}$. $\mathcal{E}_{i+1}$ may also reveal some nodes of $\mathcal{H}$ that were unknown until the i-th level. All these nodes will form a new generalized node for $\mathcal{H}_{i+1}$ (this falls into case (α)).

We next represent $Tr(\mathcal{H}_i^g)$ and $\mathcal{E}_{i+1}$ according to the new generalized nodes. If (α) or (β) is the case for all generalized nodes of $\mathcal{H}_{i+1}$, then all minimal transversals and $\mathcal{E}_{i+1}$ remain as they were. If (γ) is the case, assume that a generalized node $\mathcal{X}$ is divided into $\mathcal{X}_1$ and $\mathcal{X}_2$. Obviously, $\mathcal{E}_{i+1}$ contains only $\mathcal{X}_2$ while every minimal transversal $\mathcal{T}^g$ of $\mathcal{H}_i^g$ contains both $\mathcal{X}_1$ and $\mathcal{X}_2$. Since one of these nodes suffices for $\mathcal{T}^g$ to be a minimal hitting set of $\mathcal{H}_i^g$, two minimal transversals emerge from $\mathcal{T}^g$: one containing $\mathcal{X}_1$ and another one containing $\mathcal{X}_2$ (the generalized nodes of type (α) and (β) of $\mathcal{T}$ also appears in these minimal transversals). If $\mathcal{T}^g$ contains κ generalized nodes of type (γ), then $\mathcal{T}^g$ corresponds now to 2^κ pairwise different minimal transversals of $\mathcal{H}_i^g$, that is, all possible combinations of the two parts in which type (γ) nodes of $\mathcal{T}^g$ are divided, along with the generalized nodes of type (α) and (β) of $\mathcal{T}$. Notice that all these *offsprings* of $\mathcal{T}^g$ are not necessarily hitting sets of $\mathcal{H}_{i+1}^g$.

According to Proposition 6, $Tr(\mathcal{H}_{i+1}^g)$ is given by the relation:

$$\begin{aligned} Tr(\mathcal{H}_{i+1}^g) &= Tr(\mathcal{H}_i^g \cup \{\mathcal{E}_{i+1}^g\}) = Min(Tr(\mathcal{H}_i^g) \vee Tr(\{\mathcal{E}_{i+1}^g\})) \\ &= Min(Tr(\mathcal{H}_i^g) \vee \{\{v_\mathcal{X}\} : v_\mathcal{X} \in \mathcal{E}_{i+1}^g\}) \end{aligned} \tag{9}$$

```
for $k = 0, \ldots, m-1$ do
    Add $\mathcal{E}_{k+1}$
    Update the set of generalized nodes
    Express $Tr(\mathcal{H}_k^g)$ and $\mathcal{E}_{k+1}$ as sets of generalized nodes of level $k+1$
    Compute $Tr(\mathcal{H}_{k+1}^g) = Min(Tr(\mathcal{H}_k^g) \vee \{\{v_\mathcal{X}\} : v_\mathcal{X} \in \mathcal{E}_{i+1}^g\})$
end for
Output $Tr(\mathcal{H}_m)$
```

Algorithm 2: The modified algorithm of Berge based on generalized nodes

```
Add $\mathcal{E}_1$
Update the set of generalized nodes
Express $\mathcal{E}_1$ as set of generalized nodes
Compute $\mathcal{T} = Tr(\mathcal{E}_1)$
Call add_next_hyperedge($\mathcal{T}$,$\mathcal{E}_2$)
```

Algorithm 3: Depth-first transversal computation

Algorithm 2 is a modification of the simple scheme of Berge that computes the minimal transversals of the partial generalized hypergraphs according to Equation (9). During all intermediate steps, only the generalized transversals are kept which, in turn, are split after the addition of the next hyperedge. Experimental evaluation has shown that this dramatically reduces the number of intermediate transversals (see Example 1), especially at the early stages (where the generalized nodes are few but large) and greatly improves the time performance and the memory requirements. After the addition of the last hyperedge, Algorithm 2 outputs all minimal transversals of the input hypergraph.

Theorem 2 *Algorithm 2 correctly generates all minimal transversals of any simple hypergraph $\mathcal{H}$.*

Proof: It follows from Equation (9) and Lemma 1. □

3.3 Depth-First Transversal Computation

Although Algorithm 2 is more efficient than Algorithm 1, one still may have to wait for a long time for the first final minimal transversal to be output. This happens because it is based on a sort of *breadth-first* computation: all minimal transversals are computed after a new hyperedge is added and, after the addition of the last one, all final minimal transversals follow almost with zero delay one from the other.

Having in mind the rate of output and the memory requirements, we further improve our algorithm by implementing a *depth-first* computation of the minimal transversals: Suppose that at a certain level k we have computed a minimal transversal $\mathcal{T}$ of $\mathcal{H}_k^g$. We add the next hyperedge and determine the generalized nodes, as described above. From $\mathcal{T}$ several minimal transversals follow. However, instead of computing them all, we compute one, add the next

```
procedure add_next_hyperedge($\mathcal{T}$,$\mathcal{E}$) {
Update the set of generalized nodes
Express $Tr(\mathcal{H}^k)$ and $\mathcal{E}$ as sets of generalized nodes of level $k+1$
while generate_next_transversal($\mathcal{T}$,$\mathcal{T}'$,$l$) do
  { $\mathcal{T}'$ is the $l$-th offspring of $\mathcal{T}$ }
  if $\mathcal{E}$ is the last hyperedge then
    output $\mathcal{T}'$
  else
    { Let $\mathcal{E}'$ be the next hyperedge }
    Call add_next_hyperedge($\mathcal{T}'$,$\mathcal{E}'$)
    $l = l + 1$
  end if
end while
}
```

Procedure 4: A procedure for adding the next hyperedge

```
boolean function generate_next_transversal($\mathcal{T}$,$\mathcal{T}'$,$l$) {
if $l \leq |Min(\mathcal{T} \vee \mathcal{E})|$ then
  generate_next_transversal = true
  $\mathcal{T}'$ is the $l$-th element of the set $Min(\mathcal{T} \vee \mathcal{E})$
else
  generate_next_transversal = false
end if
}
```

Function 5: A function for computing the next minimal transversal

hyperedge and continue until all hyperedges have been added; in this case we output the final minimal transversal. We then backtrack to the previous level, pick the next minimal transversal, etc.

The whole procedure is described by Algorithm 3. At some level k, procedure `add_next_hyperedge(`$\mathcal{T}$,$\mathcal{E}$`)` (see Procedure 4) is called for adding the next hyperedge $\mathcal{E}$ to the current intermediate minimal transversal $\mathcal{T}$, which, in turn, repeatedly calls the boolean function `generate_next_transversal(`$\mathcal{T}$,$\mathcal{T}'$,l`)` (see Function 5) that returns the l-th partial minimal transversal $\mathcal{T}'$ of the new hypergraph that follows from $\mathcal{T}$. `generate_next_transversal(`$\mathcal{T}$,$\mathcal{T}'$,l`)` is called until no more minimal transversals follow from $\mathcal{T}$ after the addition of $\mathcal{E}$, in which case `generate_next_transversal()` becomes false. After a new minimal transversal $\mathcal{T}'$ is returned, `add_next_hyperedge()` is called recursively for $\mathcal{T}'$ and the next hyperedge.

The operation of Algorithm 3 resembles a preorder visit of a tree of transversals with root the single (generalized) minimal transversal of the first hyperedge, and internal nodes at some level, the minimal transversals of the partial generalized hypergraph at that level. The descendants of a minimal transversal are the minimal transversals of the next hypergraph which include this transversal.

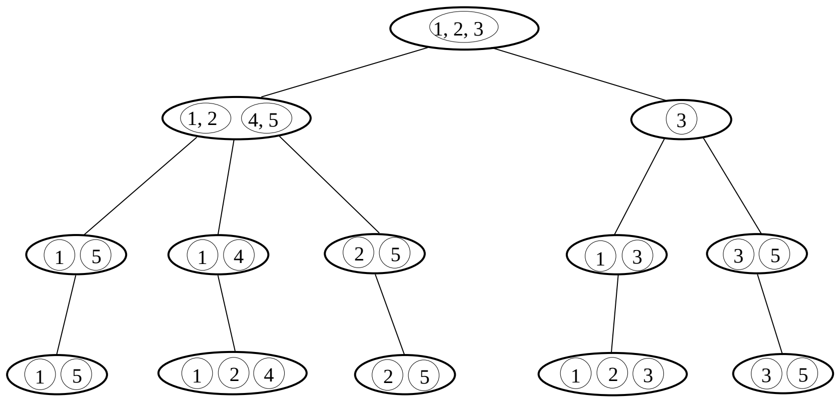

Figure 1: Transversal tree of the hypergraph $\mathcal{H} = \{\{1,2,3\}, \{3,4,5\}, \{1,5\}, \{2,5\}\}$. The tree is visited in preorder.

Finally, the leaves of the tree at level m are the minimal transversals of the original hypergraph.

Example 2 *Consider the hypergraph with 5 nodes and 4 hyperedges* $\mathcal{H} = \{\{1,2, 3\}, \{3,4,5\}, \{1,5\}, \{2,5\}\}$. *The tree of transversals which corresponds to the addition of the hyperedges according to the giver order (top to bottom) is shown in Fig. 1. Generalized nodes are denoted by circles with thin lines. For instance, a partial minimal transversal of the hypergraph consisting of the first two hyperedges is* $\{\{1,2\}, \{4,5\}\}$.

Remark. Notice that there is no need to calculate $Min(\mathcal{T} \vee \mathcal{E})$ every time the function `generate_next_transversal()` is called. Instead, in our implementation a more efficient approach was adopted which selects the split parts of the generalized nodes according to the binary expansion of l.

4 Avoiding Regenerations

Depth-first computation further improves the efficiency of our algorithm since it aims to produce the output in a uniform manner. However, regarding the space efficiency, the problem of storing every generated minimal transversal until the end of the algorithm still remains. This is because a newly generated minimal transversal may have already been generated and thus it needs to be compared with all previously generated ones. What is needed is a way of ruling out the possibility that a new minimal transversal has already been generated, or will be generated in some subsequent step, without storing all minimal transversals for comparison.

To this end, we further improve Algorithm 3 by a selective way of producing new minimal transversals. The improved algorithm assures that no regeneration occurs at any intermediate level. The advantage of this approach is that search in some subtrees stops at higher levels instead of exhaustively generating everything that would subsequently need to be compared to previous minimal transversals and, possibly, discarded. We explain the method in the sequel.

4.1 Appropriate nodes

Definition 11 *Let $\mathcal{H} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ be a hypergraph on $\mathcal{V}$ and $\mathcal{T}$ be a minimal transversal of the partial hypergraph $\mathcal{H}_k$ of $\mathcal{H}$. We say that a generalized node $v \in \mathcal{V} \setminus \mathcal{T}$ is an* appropriate node *for $\mathcal{T}$ if v is the only redundant node in the hitting set $\mathcal{T} \cup \{v\}$ for $\mathcal{H}_k$, i.e., no other node in $\mathcal{T} \cup \{v\}$ except v can be removed and the remaining set still be a hitting set of $\mathcal{H}_k$.*

Notice that an appropriate node for a minimal transversal $\mathcal{T}$ can be easily identified in polynomial time.

Let $\mathcal{H}_k$ be the partial hypergraph of the first k hyperedges defined on generalized nodes (the superscript is omitted for simplicity) and let $\mathcal{T}$ be a minimal transversal of $\mathcal{H}_k$. Suppose that $\mathcal{T}$ contains κ_α, κ_β, and κ_γ generalized nodes of type (α), (β), and (γ), respectively (defined in Subsection 3.2). As already explained, after the addition of the next hyperedge $\mathcal{E}_{k+1}$, the determination of the new set of generalized nodes results in 2^{κ_γ} minimal transversals for $\mathcal{H}_k$. Suppose that $\mathcal{T}'$ is such an offspring of $\mathcal{T}$. One may see that if $\mathcal{T}'$ contains at least one node of type (β) (that is, a generalized node that appears in both $\mathcal{T}'$ and $\mathcal{E}_{k+1}$), then $\mathcal{T}'$ is a hitting set of $\mathcal{H}_{k+1}$. If it does not, then $\mathcal{T}'$ has to be augmented by a node of $\mathcal{E}_{k+1}$ to ensure that the resulting set of nodes is a hitting set of $\mathcal{H}_{k+1}$. This procedure would result in $|\mathcal{E}_{k+1}|$ hitting sets (from $\mathcal{T}'$), not necessarily minimal ones, and has to be repeated for every offspring of $\mathcal{T}$ (This is the Cartesian product in Berge's scheme). All these potentially minimal transversals have to be stored and the $Min()$ operator must be applied in a way that only the minimal ones remain and all duplicates are removed.

The above procedure can be very demanding regarding its time and space requirements. Here is where the notion of appropriate node comes into play. Using this notion, we may compute the Cartesian product of the offsprings of $\mathcal{T}$ by $\mathcal{E}_{k+1}$ in a way that excludes the possibility of regenerating a minimal transversal of $\mathcal{H}_{k+1}$. In addition, every resulting hitting set is always minimal and, hence, neither the $Min()$ operator needs to be applied, nor this hitting set has to be stored to be compared with the next ones. Thus, the computational effort and the space requirements are reduced and the algorithm becomes more efficient. We explain the whole procedure by distinguishing two cases:

Case A *There exist at least one generalized node of type (β) in $\mathcal{T}$.* In this case, every offspring of $\mathcal{T}$ (that is, 2^{κ_γ} in total) is a minimal transversal of $\mathcal{H}_{k+1}$. If $\mathcal{T}_i$ is one of these ($i = 1, \ldots, 2^{\kappa_\gamma}$), then $\mathcal{T}_i$ hits all hyperedges of $\mathcal{H}_k$ ($\mathcal{T}$ is a minimal transversal of $\mathcal{H}_k$) and $\mathcal{E}_{k+1}$ (due to the existence of at least one generalized node of type (β)), too. Moreover, $\mathcal{T}_i$ is minimal since the removal of any of its nodes would result in a set that does not hit $\mathcal{H}_k$. This also holds even for the offspring, say $\mathcal{T}_0$, that contains the κ_γ split parts of the generalized nodes of type (γ) of $\mathcal{T}$ that are contained only in $\mathcal{T}$ and not in $\mathcal{E}_{k+1}$.

To sum up, the existence of at least one generalized node of type (β) in $\mathcal{T}$ results in 2^{κ_γ} minimal transversals for $\mathcal{H}_{k+1}$. This holds even if $\kappa_\alpha = 0$ (these nodes just participate so that each offspring hits $\mathcal{H}_k$), or $\kappa_\gamma = 0$ (only 2^0 minimal transversals emerge, the one containing the generalized nodes of type (β)). Thus, there always exists at least one descendant of $\mathcal{T}$ in the transversal tree. These minimal transversals are pairwise different and no other offspring of $\mathcal{T}$ that is minimal transversal of $\mathcal{H}_{k+1}$ exists. We have, thus, proved the following lemma:

Lemma 2 *After the addition of $\mathcal{E}_{k+1}$, if a minimal transversal $\mathcal{T}$ of $\mathcal{H}_k$ contains at least one generalized node of type (β), then $\mathcal{T}$ gives exactly 2^{κ_γ} pairwise different minimal transversals for $\mathcal{H}_{k+1}$, where κ_γ is the number of the generalized nodes of type (γ) of $\mathcal{T}$.*

Case B *No generalized node of type (β) is contained in $\mathcal{T}$.* In this case, every offspring $\mathcal{T}_i$ of $\mathcal{T}$ is a minimal hitting set of $\mathcal{H}_{k+1}$ except from $\mathcal{T}_0$ (the one that contains the κ_γ split parts of the generalized nodes of type (γ) of $\mathcal{T}$ that are contained only in $\mathcal{T}$ and not in $\mathcal{E}_{k+1}$), since it does not hit $\mathcal{E}_{k+1}$. Thus, $2^{\kappa_\gamma} - 1$ minimal transversals of $\mathcal{H}_{k+1}$ emerge from $\mathcal{T}$, while $\mathcal{T}_0$ may also result in a hitting set of $\mathcal{H}_{k+1}$, if it is augmented by a node of $\mathcal{E}_{k+1}$. Instead, however, of adding each node of $\mathcal{E}_{k+1}$ in turn and outputting $|\mathcal{E}_{k+1}|$ hitting sets, we only add each node of $\mathcal{E}_{k+1}$ in turn that is *appropriate* for $\mathcal{T}$ (and, hence, for $\mathcal{T}_0$). If $v \in \mathcal{E}_{k+1}$ is an appropriate node for $\mathcal{T}$, then the set $\mathcal{T} \cup \{v\}$ is a minimal transversal of $\mathcal{H}_{k+1}$ (Notice that a new generalized node of $\mathcal{E}_{k+1}$ that was not included in any of the first k hyperedges, is also an appropriate node for $\mathcal{T}$). By this way, $|appr(\mathcal{T}, \mathcal{E}_{k+1})|$ minimal transversals for $\mathcal{H}_{k+1}$ also emerge from $\mathcal{T}$, where $appr(\mathcal{T}, \mathcal{E}_{k+1})$ is the set of the appropriate nodes of $\mathcal{T}$ contained in $\mathcal{E}_{k+1}$. All these $2^{\kappa_\gamma} - 1 + |appr(\mathcal{T}, \mathcal{E}_{k+1})|$ in total minimal transversals are pairwise different.

It is also possible a subset of $\mathcal{T} \cup \{v\}$, where $v \in \mathcal{E}_{k+1}$ is a non appropriate node for $\mathcal{T}$, to be a minimal transversal of $\mathcal{H}_{k+1}$. As the next lemma states, all these minimal transversals are not lost (actually, they will be produced at the same level of computation):

Lemma 3 *Let $\mathcal{T}$ be a minimal transversal of $\mathcal{H}_k$ and $v \in \mathcal{E}_{k+1}$ be a non appropriate node of $\mathcal{T}$. Let also $\mathcal{T}_0$ be the offspring of $\mathcal{T}$, after the addition of $\mathcal{E}_{k+1}$, that follows from the split parts of the generalized nodes of type (γ) of $\mathcal{T}$ that are not contained in $\mathcal{E}_{k+1}$. Then, each minimal transversal of $\mathcal{H}_{k+1}$ that is subset of $\mathcal{T}_0 \cup \{v\}$, emerges from some other minimal transversal $\mathcal{T}'$ of $\mathcal{H}_k$ according to Case A.*

Proof: Since v is not appropriate for $\mathcal{T}$, at least one node of $\mathcal{T}$ may be removed and the resulting subset $\mathcal{T}'$ of $\mathcal{T} \cup \{v\}$ still remains a hitting set of $\mathcal{H}_k$. Since v appears in both $\mathcal{E}_{k+1}$ and $\mathcal{T}'$, all minimal transversals of $\mathcal{H}_{k+1}$ that are offsprings of $\mathcal{T}'$ will emerge, according to Case A. $\square$

Hence, the following lemma has been shown:

```
boolean function generate_next_transversal($\mathcal{T}, \mathcal{T}', l$) {
if $\kappa_\beta \neq 0$ then
  if $l \leq 2^{\kappa_\gamma}$ then
    generate_next_transversal = true
    $\mathcal{T}'$ is the $l$-th offspring of $\mathcal{T}$
  else
    generate_next_transversal = false
  end if
else if $\kappa_\beta = 0$ then
  if $l \leq 2^{\kappa_\gamma} - 1$ then
    generate_next_transversal = true
    $\mathcal{T}'$ is the $l$-th offspring of $\mathcal{T}$ (except $\mathcal{T}_0$)
  else if $2^{\kappa_\gamma} \leq l \leq 2^{\kappa_\gamma} - 1 + |appr(\mathcal{T}, \mathcal{E})|$ then
    generate_next_transversal = true
    $\mathcal{T}'$ is the union of $\mathcal{T}_0$ by the $(l - 2^{\kappa_\gamma} + 1)$ node of $\mathcal{T}$ the set $appr(\mathcal{T}, \mathcal{E})$
  else
    generate_next_transversal = false
  end if
end if
}
```

Function 6: The enhanced function for computing the next minimal transversal

Lemma 4 *After the addition of $\mathcal{E}_{k+1}$, if a minimal transversal $\mathcal{T}$ of $\mathcal{H}_k$ contains no generalized node of type (β), then $\mathcal{T}$ gives exactly $2^{\kappa_\gamma} - 1 + |appr(\mathcal{T}, \mathcal{E}_{k+1})|$ pairwise different minimal transversals for $\mathcal{H}_{k+1}$, where κ_γ is the number of the generalized nodes of type (γ) of $\mathcal{T}$ and $appr(\mathcal{T}, \mathcal{E}_{k+1})$ is the set of the appropriate nodes of $\mathcal{T}$ contained in $\mathcal{E}_{k+1}$.*

4.2 The Improved Algorithm

All ideas described in the previous section were implemented and were incorporated in Function 6. Algorithm 3 along with Procedure 4 and the enhanced function `generate_next_transversal()` (see Function 6) constitute the final version of our algorithm. Function 6 differs from Function 5 in that it adds to $\mathcal{T}$ only the nodes of $\mathcal{E}_{k+1}$ that are appropriate for $\mathcal{T}$, leaving out any non appropriate ones. The correctness of the algorithm is established by the following theorem:

Theorem 3 *The proposed algorithm correctly generates all minimal transversals of a simple hypergraph $\mathcal{H}$ without regenerations, in space polynomially bounded by the size of the input hypergraph.*

Proof: The proof follows by induction: Assume that at the k-th level, all minimal transversals of $\mathcal{H}_k$ have been correctly generated. We will show now that after the addition of $\mathcal{E}_{k+1}$, the algorithm correctly generates all minimal transversals of $\mathcal{H}_{k+1}$ without regenerations.

It follows from Lemmata 2 and 4 that all minimal transversals of $\mathcal{H}_{k+1}$ are generated. It remains to be shown that are all distinct from each other.

Notice that all minimal transversals of $\mathcal{H}_{k+1}$ that are offsprings of any minimal transversal of $\mathcal{H}_k$ are pairwise different, since each consists of a part from every generalized node of its ancestor. Those that are offsprings of the same ancestor are pairwise different as explained in Lemmata 2 and 4. Observe that these minimal transversals of $\mathcal{H}_{k+1}$ are also minimal for $\mathcal{H}_k$, as well. This establishes that they all also differ from any minimal transversal of $\mathcal{H}_{k+1}$ that is produced with the addition of an appropriate node, since the latter is not minimal for $\mathcal{H}_k$. Therefore, the only remaining case to be checked is between two minimal transversals that were produced by the addition of an appropriate node and have different ancestors (Those that have the same ancestor are different, as explained in Lemma 4). Let $t = \mathcal{T}_0 \cup \{v\}$ and $t' = \mathcal{T}'_0 \cup \{v'\}$ be two minimal transversals of $\mathcal{H}_{k+1}$ that were produced by the addition of the appropriate nodes v and v' of $\mathcal{E}_{k+1}$, respectively. Assume to the contrary that $t = t'$. If $v = v'$ then $\mathcal{T}_0 = \mathcal{T}'_0$, which means that t and t' are both descendants of the same minimal transversal of $\mathcal{H}_k$, a contradiction by Lemma 4. For the same reason, $\mathcal{T}_0 \neq \mathcal{T}'_0$. Thus, v (respectively v') must be a node of $\mathcal{T}'_0$ (respectively $\mathcal{T}_0$). Hence, both $\mathcal{T}_0$ and $\mathcal{T}'_0$ are hitting sets of $\mathcal{H}_{k+1}$ since each includes a node of $\mathcal{E}_{k+1}$, again a contradiction. Hence, all minimal transversals that the algorithm produces at level $(k+1)$ are pairwise different.

Regarding the space complexity of the algorithm, as regeneration is not allowed, no minimal transversal has to be stored at any level of the tree and, hence, the space requirements are determined by the total size of the activation records of the function `add_next_hyperedge()` in all its recursive calls. If m is the number of the hyperedges of the input hypergraph, then at most m activation records are stored at any point of the computation (since the space of an activation record is reused after the end of its call). In addition, `add_next_hyperedge()` is called by the function `generate_next_transversal()` (see Function 6). The formal parameters of `generate_next_transversal()` are the transversal $\mathcal{T}$, its l-th descendant $\mathcal{T}'$ at the computation tree, plus the number l itself. Since the transversal hypergraph can have up to 2^n hyperedges, l can be stored using at most n bits. Also, determining whether a node is appropriate for $\mathcal{T}$ requires polynomial space as well. It follows that the activation record of `generate_next_transversal()` requires only polynomial space and, consequently, the recursive calls of `add_next_hyperedge()` require polynomial space as well. $\square$

5 Experimental Evaluation

Implementation and experimental evaluation of our algorithm were carried out on a 2.8MHz Pentium V PC with 512MB RAM, running Linux (Mandrake 10). The main part of the code consists of the subroutines `add_next_hyperedge()` and `generate_next_transversal()` (see Procedure 4 and Function 6, respectively). The main difference of the current implementation compared to the one in [29] is the more sophisticated way to search for appropriate nodes. This required the redesign of some auxiliary data structures and resulted in an increase of the size of the code by more than 50%. However this payed off in the performance of the algorithm. A linux-executable is available in `http://lca.ceid.upatras.gr/∼estavrop/transversal`.

Additional algorithms were also implemented to support our experiments, to verify the correctness of our code and for comparison reasons. The algorithm of Berge was firstly implemented, in order to evaluate our ideas and to compare with in the sequel. As already discussed, the simple scheme of Berge is both time and memory demanding and thus, it can by applied only for small instances. To verify the correctness of the code for relatively large instances, we used the *duality property* (see Corollary 1). The duality property also offers the possibility to evaluate the algorithm on instances with specific properties (very large number of hyperedges and very small number of transversals).

We have also implemented a simple scheme that computes the transversal hypergraph exhaustively: it generates all possible hitting sets of the input hypergraph (which are of the order of 2^n) and subsequently checks whether each of them is minimal. Naturally, both memory requirements and time performance of this simple algorithm are unacceptable.

To compare with, we considered recently implemented algorithms for the problem. Except our algorithm (KS), we are aware of only two implementations of algorithms for this problem: the algorithm of Fredman and Khachiyan as implemented by Boros at al. in [6][1] (BEGK) and the algorithm of Bailey et al. given in [1][2] (BMR). For each algorithm, we tested total CPU time (in seconds) and total memory (in megabytes) required for every test case.

All algorithms were first evaluated on certain test cases described in [6]:

- Matching graph ($\mathcal{M}(n)$): a graph with n nodes (n is even) and $n/2$ edges forming an induced matching. This graph type has a small number of edges but a large number of minimal transversals (namely, $2^{n/2}$).

- Dual Matching graph ($\mathcal{DM}(n)$): the dual graph of $\mathcal{M}(n)$. This graph type has a large number of edges but a small number of minimal transversals.

- Threshold graph ($\mathcal{TH}(n)$): a graph with n nodes (n is even) labelled from 1 to n, and edge set $\{\{i,j\} : 1 \le i < j \le n, j \text{ is even}\}$. This graph type has a small number of edges (namely, $n^2/4$) and a small number of minimal transversals (namely, $n/2+1$).

- Self-Dual Threshold graph ($\mathcal{SDTH}(n)$): the self-dual hypergraph $\mathcal{H}$ with n nodes obtained from $\mathcal{TH}(n)$ and $\mathcal{DTH}(n)$ as follows:

  $$\mathcal{H} = \{\{n-1,n\}\} \cup \{\{n-1\} \cup \mathcal{E} | \mathcal{E} \in \mathcal{TH}(n-2)\} \cup \{\{n\} \cup \mathcal{E} | \mathcal{E} \in \mathcal{DTH}(n-2)\}.$$

  This hypergraph has a polynomially bounded number of hyperedges (and, minimal transversals), namely, $(n-2)^2/4 + n/2 + 1$.

[1] An executable was downloaded from `http://paul.rutgers.edu/~elbassio/dual.html`.
[2] An executable was provided by the authors.

Table 1: Experimental evaluation on Matching and Dual Matching graphs.

	CPU time (seconds)			Memory (MB)		
$\mathcal{M}(n)$	BEGK	BMR	KS	BEGK	BMR	KS
20	0.74	0.01	0.00	1	1	1
24	2.1	0.04	0.01	15	1	1
28	9.4	0.2	0.04	17	2	1
30	30	0.43	0.07	17	3	1
32	74	0.85	0.17	20	5	1
34	174	1.76	0.38	37	12	1
36	372	5.6	0.81	44	13	1
38	964	12	1.71	85	27	1
40	1196	14	3.66	94	75	1
$\mathcal{DM}(n)$	BEGK	BMR	KS	BEGK	BMR	KS
20	0.29	0.66	0.02	13	1	1
24	2.09	3.9	0.33	15	1	1
28	21	53	1.4	18	3	6
30	72	210	4.5	19	6	11
32	252	860	16	26	13	22
34	911	2360	57	34	26	45
36	2188	12463	197	44	72	89
38	8756	36600	655	88	139	178
40	35171	201142	2167	189	464	357

- Self-Dual Fano-Plane graph ($\mathcal{SDFP}(n)$): a graph with n nodes and $(k-2)^2/4+k/2+1$ hyperedges, where $k=(n-2)/7$. To construct it, we start with the hypergraph $\mathcal{H}_0 = \{\{1,2,3\},\{1,5,6\},\{1,7,4\},\{2,4,5\},\{2,6,7\},\{3,4,6\},\{3,5,7\}\}$ (that represents the set of lines in a Fano plane and is self-dual) and we set $\mathcal{H}=\mathcal{H}_1\cup\mathcal{H}_2\cup\ldots\cup\mathcal{H}_k$, where $\mathcal{H}_1,\mathcal{H}_2,\ldots,\mathcal{H}_k$ are k disjoint copies of $\mathcal{H}_0$. The dual graph of $\mathcal{H}$ is the hypergraph of all 7^k unions obtained by taking one hyperedge from each of the k copies of $\mathcal{H}_0$,. We finally define $\mathcal{SDFP}(n)$ as the hypergraph obtained by self-dualizing $\mathcal{H}$, as we did for the threshold graphs.

Comparison results on these test cases are summarized in Tables 1 and 2. Problem sizes are identified by the number of nodes, n. For each test case we report the total CPU time, in seconds, and the total memory, in megabytes, required for each algorithm to generate all minimal transversals of the specified hypergraphs.

From these tables it is immediate that the KS algorithm outperforms, with respect to time, the other two in all test cases, with the BMR algorithm being the second fastest in most test cases. Regarding the memory requirements, the KS algorithm uses almost zero memory in those cases where the size of the input graph is much smaller than the output graph (matching graph). In cases where the input graph has many edges, memory requirements of the KS algorithm

Table 2: Experimental evaluation on Threshold, Self-Dual Threshold, and Self-Dual Fano-Plane graphs.

	CPU time (seconds)			Memory (MB)		
$\mathcal{TH}(n)$	BEGK	BMR	KS	BEGK	BMR	KS
40	0.33	0.14	0.02	21	1	1
60	0.61	1.24	0.16	29	1	1
80	1.96	6.5	0.65	36	2	2
100	4.6	24	1.9	44	2	5
120	10	72	4.9	52	3	8
140	22	194	11	60	5	12
160	40	460	23	67	7	17
180	75	1000	44	77	8	24
200	289	1968	82	85	12	33
$\mathcal{SDTH}(n)$	BEGK	BMR	KS	BEGK	BMR	KS
42	0.76	0.14	0.12	21	1	1
62	4.56	1.28	0.66	29	1	1
82	18	6.6	2.25	36	1	2
102	55	24	6.2	44	2	5
122	156	73	14	52	3	8
142	453	193	30	60	5	12
162	1125	458	56	69	7	18
182	1859	1004	101	77	11	26
202	3643	1976	178	89	11	34
$\mathcal{SDFP}(n)$	BEGK	BMR	KS	BEGK	BMR	KS
16	0.11	0.01	0.00	11	1	1
23	2.05	0.26	0.06	15	1	1
30	62	12	1.54	16	1	1
37	2130	553	63	22	8	6

are comparable to the other two algorithms. This is because the KS algorithm needs to build auxiliary data structures that store all intermediate transversals from the root to a leaf of the transversal tree, that is, its memory requirements are proportional to the size of the input hypergraph as proved in Theorem 3. As a general characteristic, the proposed algorithm performs better in time and space in problem instances where the input hypergraph has fewer hyperedges than the output.

This last observation is also verified in Table 3, where the performance in three test cases, especially unfavorable for our algorithm, is reported. These test cases come from the area of data mining and specifically from discovering emerging patterns in large datasets, a problem that reduces to the generation of all minimal transversals of a hypergraph. These datasets were used in [1] and provided to us by the authors. In Table 3, instance parameters indicate the number of nodes, n, the number of hyperedges, m, and the number of minimal transversals, t. For each test case we report the total CPU time, in seconds,

Table 3: Experimental evaluation on large datasets.

Instance Parameters			CPU time			Memory		
n	m	t	BEGK	BMR	KS	BEGK	BMR	KS
287	48226	97	1332	1241	1648	161	16	53
	92699	99	4388	4280	6672	208	33	105
	108721	99	5898	7238	9331	209	41	122

and the total memory, in megabytes, required for the algorithms to generate the output. As shown in Table 3, input hypergraphs have many hyperedges and few minimal transversals. In these runs the memory requirements of our algorithm are greater than those of the BMR as the algorithm needs to maintain some data structures that hold a whole path in the transversal tree. This path is very long in these instances. Also, the time performance of the BEGK algorithm is notable here.

Finally, we tested all algorithms on a common set of randomly generated hypergraphs. A random hypergraph generator was implemented and used for this task. Given the number of nodes, n, and the desired number of hyperedges, m, the random hypergraph generator uniformly and independently generates m sets of nodes, each of them corresponding to a hyperedge of the instance. The cardinality of each set lies between 1 and n, while a node belongs to a hyperedge with probability between p_l and p_u, $0 \leq p_l \leq p_u \leq 1$. Care was taken that the produced hypergraph be simple, that is, that no hyperedge is fully included in another one.

Very few, indicative results are shown in Table 4. Problem sizes were identified by the number of nodes, n, and the number of hyperedges, m, of the hypergraph, while p_l and p_u are the lower and upper probability bounds, respectively, for a node to participate to a hyperedge. The number of minimal transversals, t, also characterizes the size of the problem. Reports are averages over 30 different runs for each instance size.

Table 4: Experimental evaluation on random instances. Reports are averages over 30 runs.

Instance Parameters					CPU time (seconds)			Memory (MB)		
n	m	p_l	p_u	t	BEGK	BMR	KS	BEGK	BMR	KS
50	100	0.5	0.9	150000	616	3.7	6.7	27	7	1
50	100	0.5	0.7	300000	1478	7.3	11.8	87	12	1
50	100	0.4	0.6	1.8×10^6	–	73	84	–	90	1
60	500	0.5	0.9	6.7×10^6	4900	420	970	266	313	1
60	500	0.5	0.7	27×10^6	–	–	4600	–	–	1
60	500	0.4	0.6	330×10^6	–	–	120000	–	–	1

For these instances the results vary greatly as for each one of the three algorithms there are instances for which it is the fastest of the three. There is large deviation even among the 30 runs with the same parameters which are averaged. This is an indication that the problem has more complicated structure that cannot be totally captured by the number of nodes and the number of edges of the input graph. As a typical behavior however, we could say that in these instances the BMR algorithm is the faster while our algorithm is the most efficient in memory requirements. For some large instances, this advantage makes it the only algorithm that can terminate within the available memory and generate the whole transversal hypergraph. For these instances the BMR and BEGK algorithms suffered from memory starvation (shown by a dash in Table 4) while the KS algorithm produced the whole hypergraph using less than 1MB of memory. For problem instances like this, care was taken to split the output into several files as the system-restricted maximum file size was not enough to store the whole output.

6 Conclusions

In this paper we presented an algorithm for solving the Transversal Hypergraph Generation problem. This problem may produce a large output and therefore algorithms for solving it must be evaluated with respect to both their running time and memory requirements. We prove that our algorithm produces the output without regenerations in space that is proportional to the size of the input hypergraph. We are not aware of any other algorithm that achieves input–polynomial space bound. Our algorithm operates in a generate-and-forget mode and no output bit is needed to be stored for further manipulation. This property makes it suitable for solving efficiently instances with small input size and large output. We have also presented experimental evaluation of the algorithm and compare it with other known algorithms.

Further research includes a theoretical investigation of the time complexity of the algorithm and specifically the delay between consecutive outputs. Future work also includes fine-tuning of the code to further improve its performance.

Acknowledgments

We thank Thomas Manoukian for providing the executable for the algorithm in [1] and the datasets reported in Table 3.

References

[1] J. Bailey, T. Manoukian, and K. Ramamohanarao. A fast algorithm for computing hypergraph transversals and its application in mining emerging patersn. In *Proc. of the 3rd IEEE International Conference on Mining (ICDM 2003)*, pages 485–488. IEEE Computer Society, December 2003.

[2] R. Ben-Eliyahu and R. Dechter. On computing minimal models. *Annals of Mathematics and Artificial Intelligence*, 18:3–27, 1996.

[3] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45 of *North Holland Mathematical Library*. Elsevier Science Publishers B.V., Amsterdam, 1989.

[4] J. C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, 123:50–63, 1995.

[5] B. Bollobás. *Combinatorics: Set Systems, Hypergraphs, Families of Vectors and Combinatorial Probability*. Cambridge University Press, Great Britain, 1986.

[6] E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals. In *Proc. of the 11th European Symposioum on Algorithms (ESA 2003)*, volume 2432 of *LNCS*, pages 556–567. Springer, 2003.

[7] N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer Systems and Sciences*, 52:421–433, 1996.

[8] M. Cadoli. The complexity of model checking for circumscriptive formulae. *Information Processing Letters*, 44(3):113–118, 1992.

[9] Z.-Z. Chen and S. Toda. The complexity of selecting maximal solutions. *Information and Computation*, 119:231–239, 1995.

[10] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnosis and systems. *Artificial Intelligence*, 56:197–222, 1992.

[11] J. Diaz and J. Torán. Classes of bounded nondeterminism. *Mathematical Systems Theory*, 23:21–32, 1990.

[12] C. Domingo, N. Mishra, and L. Pitt. Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries. *Machine Learning*, 37(1):88–110, 1999.

[13] J.-G. Dumas, F. Heckenbach, D. Saunders, and V. Welker. Computing simplicial homology based on efficient Smith Normal Form algorithms. In *Algebra, Geometry, and Software Systems*, pages 177–206, 2003.

[14] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal of Computing*, 24(6):1278–1304, December 1995.

[15] T. Eiter and G. Gottlob. Hypergraph transversal computation and related problems in Logic and AI. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA 2002)*, volume 2424 of *LNCS/LNAI*, pages 549–564. Springer, 2002.

[16] T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. In *Proc. of the 34th ACM Symposium on Theory of Computing (STOC 2002)*, pages 14–22, Montreal, Quebec, Canada, May 2002.

[17] T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM J. Computing*, 32(2):514–537, 2003.

[18] M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21:618–628, 1996.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[20] L. A. Goldberg. *Efficient Algorithms for Listing Combinatorial Structures.* BCS Distinguished Dissertations in Computer Science. Cambridge University Press, Great Britain, 1993.

[21] J. Goldsmith, M. A. Levy, and M. Mundhenk. Limited nondeterminism. *ACM SIGACT News*, 27(2):20–29, June 1996.

[22] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, June 2003.

[23] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proc. of Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 209–216, Tucson, Arizona, USA, May 1997. ACM Press.

[24] V. Gurvich and L. Khachiyan. On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Discrete Applied Mathematics*, 96–97:363–373, 1999.

[25] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, March 1988.

[26] D. Kavvadias, C. H. Papadimitriou, and M. Sideri. On Horn envelopes and hypergraph transversals. In *Proc. of 4th Annual International Symposium on Algorithms and Computation (ISAAC'93)*, volume 762 of *LNCS*, pages 399–405, Hong Kong, 1993. Springer.

[27] D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM Journal of Computing*, 28(1):152–163, 1998.

[28] D. J. Kavvadias, M. Sideri, and E. C. Stavropoulos. Generating all maximal models of a Boolean expression. *Information Processing Letters*, 74(3-4):157–162, May 2000.

[29] D. J. Kavvadias and E. C. Stavropoulos. Evaluation of an algorithm for the transversal hypergraph problem. In J. S. Vitter and C. D. Zaroliagis, editors, *Proc. of 3th International Workshop on Algorithm Engineering (WAE99)*, volume 1668 of *LNCS*, pages 72–84, London, UK, 1999. Springer-Verlag.

[30] D. J. Kavvadias and E. C. Stavropoulos. Monotone Boolean dualization is in co-NP[$\log^2 n$]. *Information Processing Letters*, 85(1):1–6, January 2003.

[31] C. M. R. Kintala and P. Fischer. Refining nondeterminism in relativized polynomial-time bounded computations. *SIAM J. Computing*, 9:46–53, 1980.

[32] E. L. Lawler. Covering problems: duality relations and a new method of solution. *SIAM Journal of Applied Mathematics*, 14(5):1115–1132, September 1966.

[33] H. Mannila and K.-J. Räihä. Design by example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 33:126–141, 1986.

[34] C. H. Papadimitriou. NP-completeness: A retrospective. In *Proc. of 24th International Colloquium on Automata, Languages, and Programming*, volume 1256 of *LNCS*, pages 2–6, Bologna, Italy, July 1997.

[35] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[36] S. Sarkar and N. Savarajan. Hypergraph models for cellular mobile communication systems. *IEEE Transactions on Vehicular Technology*, 47(2):460–471, May 1998.

[37] B. Selman and H. A. Kautz. Model-preference default theories. *Artificial Intelligence*, 45:287–322, 1990.

[38] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

[39] B. Zanuttini. Approximation of relations by propositional formulas: Complexity and semantics. In S. Koenig and R. Holte, editors, *Proc. of the SARA 2002*, volume 2371 of *LNCS*, pages 242–255. Springer-Verlag, 2002.