IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 28, 2018, accepted January 21, 2019, date of publication January 25, 2019, date of current version February 22, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2895296

Symbolic Learning for Improving the Performance of Transversal-Computation Algorithms

VÍCTOR IVÁN GONZÁLEZ-GUEVARA[1], SALVADOR GODOY-CALDERON[1], EDUARDO ALBA-CABRERA[2], AND HIRAM CALVO[1]

[1]Centro de Investigación en Computación, Instituto Politécnico Nacional, Mexico City 07738, Mexico
[2]Colegio de Ciencias e Ingenierías, Universidad San Francisco de Quito, Quito 170901, Ecuador

Corresponding author: Hiram Calvo (hcalvo@cic.ipn.mx)

This work of V. I. González-Guevara, S. Godoy-Calderon, and H. Calvo was supported in part by the Consejo Nacional de Ciencia y Tecnología. (National Council for Science and Technology)-Sistema Nacional de Investigadores (National Research System), in part by the Instituto Politécnico Nacional (Comisión de Operación y Fomento de Actividades Académicas and Sistema de Becas de Exclusividad), and grants SIP 20180801, 20182114, 20195886, and 20196094, and in part by Estímulos al Desempeño a la Investigación. The work of E. Alba-Cabrera was supported by the Chancelor Grant Universidad San Francisco de Quito under Grant 2268 2015.

ABSTRACT Using the hypergraphs as the central data structure in dynamic discrete association problems is a common practice. The computation of minimal transversals (i.e., the family of all minimal hitting sets) in those hypergraphs is a well-studied task associated with a large number of practical applications. However, both the dynamic nature of the problems and the non-polynomial behavior of all currently known algorithms for that task justify the search for performance optimizations that allow transversal-computation algorithms to consistently handle the potentially large problems while optimizing the use of computational resources. This scenario has been extensively studied from the perspective of the hypergraph, rough set, and testor theories, but this paper presents the first glimpse into a symbolic learning approach. We present a symbolic learning strategy, for the class of transversal-computation algorithms, designed to guide and optimize the search process. Since the proposed strategy is based on the background knowledge about the search space and not on a specific search technique, it can be adapted to a wide variety of algorithms. We present the learning strategy as well as its adaptation into two representative transversal-computation algorithms. The comparative experimental results reveal its computational behavior on different problem families.

INDEX TERMS Symbolic learning, learning strategy, transversal hypergraph, minimal hitting set.

I. INTRODUCTION

When faced with discrete association problems, the most commonly used data structures are hypergraphs. Hypergraphs allow even fast-changing dynamic phenomena to be modeled by using sets of vertices and edges that continuously grow in time [1]. Such scenarios can be found in diverse application fields such as computational biology, social network analysis, biochemical molecule testing, data mining, distributed systems, artificial intelligence, natural language processing, etc.

Finding the complete family of minimal hitting sets in those hypergraphs is commonly referred to as the *transversal generation problem* or sometimes just as the TransHyp problem [2]. This is a common and well known task that has been studied, not only from the perspective of hypergraph theory, but from the perspective of all MONET-equivalent problems (e.g. prime implicants in boolean functions, typical testors, reducts in rough sets, etc.) [1], [3]–[5].

At some abstraction level, all algorithms for computing minimal hitting sets perform a search within their input hypergraph looking for sets of vertices that satisfy specific conditions. However, at a lower abstraction level the search mechanic strongly depends on the representation formalism used to describe the input hypergraph. Very often, the input hypergraph is represented only by its incidence matrix, which is a two dimensional binary matrix showing the membership of vertices (columns) to hyperedges (rows). A great majority of algorithms that search for minimal hitting sets, upon receiving such input matrix, will select one or more vertex subsets and test if they are hitting sets and if they are minimal. Once the tests are completed, this newly acquired knowledge is combined with some background knowledge in order to

The associate editor coordinating the review of this manuscript and approving it for publication was Zhipeng Cai.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

decide which other subsets should be tested next. Structural properties of the search space, as well as of the particular input hypergraph constitute the background knowledge and the result of each test performed to selected vertex subsets conform the acquired knowledge [6]. The synthetic accumulation of acquired knowledge and its combination with selected background knowledge allows the definition of a learning strategy for optimizing the search. Such a strategy can be adapted to work alongside many minimal hitting set-computation algorithms [7].

In this research we propose a symbolic learning strategy to guide the search for minimal hitting sets in hypergraphs. The proposed strategy is general enough to be adapted to a wide variety of existing algorithms, even when they do not follow the same search approach. First, the taxonomy of transversal-computation algorithms is reviewed, then the proposed learning strategy is explained and the performance gain of adding it to classic algorithms is experimentally tested. Finally, some conclusions are drawn from the process and future research lines are outlined.

II. THEORETICAL FRAMEWORK

Definition 1 (Hypergraph): 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 objects, and $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ is a covering of $\mathcal{V}$, that is, a non-empty and exhaustive family of subsets of $\mathcal{V}$, (e.g. $\mathcal{E}_i \neq \emptyset$ $(i = 1, \ldots, m)$ and $\bigcup_{i=1}^{m} \mathcal{E}_i = \mathcal{V}$).

The elements of $\mathcal{V}$ are called *vertices*, while the elements of $\mathcal{E}$ are called *edges* or *hyperedges*. We say that vertex $v_i \in \mathcal{V}$ *hits* edge $\mathcal{E}_j \in \mathcal{E}$ when $v_i \in \mathcal{E}_j$.

When the edges in a hypergraph conform a *Sperner* family (e.g. where no set includes any other), then the hypergraph is referred to as a *simple hypergraph,bergebook*, and by $Min(\mathcal{H})$ we denote the simple hypergraph consisting of all the minimal edges of $\mathcal{H}$ with respect to set inclusion.

Definition 2 (Hitting Set): Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A subset $\tau \subseteq \mathcal{V}$ is called a *hitting set* in $\mathcal{H}$, if it intersects all edges, that is, when $(\forall \mathcal{E}_i \in \mathcal{E})$ $[\tau \cap \mathcal{E}_i \neq \emptyset]$. A hitting set is *minimal* if none of its proper subsets is also a hitting set in $\mathcal{H}$.

Definition 3 (Transversal Hypergraph): A hypergraph $\mathcal{H}'$ is called the *transversal* (sometimes called the *dual*) of another hypergraph $\mathcal{H}$, if both are defined over the same set of vertices, and the set of edges in $\mathcal{H}'$ is the complete family of minimal hitting sets in $\mathcal{H}$. By $Tr(\mathcal{H})$ we denote the transversal or dual hypergraph of $\mathcal{H}$.

The above condition holds in both ways for simple hypergraphs, when $\mathcal{H}' = Tr(\mathcal{H})$, then necessarily $\mathcal{H} = Tr(\mathcal{H}')$. This dual relation implies that, for simple hypergraphs, $\mathcal{H} = Tr(Tr(\mathcal{H}))$, justifying that two hypergraphs $\mathcal{H}$ and $\mathcal{H}'$ are said to be dual, if they are mutually transversal of each other.

Definition 4 (Combination Operators): For two simple hypergraphs $\mathcal{H}_1 = (\mathcal{V}_1, \{f_1, f_2, \ldots, f_{m_1}\})$ and $\mathcal{H}_2 = (\mathcal{V}_2, \{g_1, g_2, \ldots, g_{m_2}\})$ there are two combination operators:

1) $\mathcal{H}_1 \cup \mathcal{H}_2 = (\mathcal{V}_1 \cup \mathcal{V}_2, \{f_1, f_2, \ldots, f_{m_1}, g_1, g_2, \ldots, g_{m_2}\})$
2) $\mathcal{H}_1 \vee \mathcal{H}_2 = (\mathcal{V}_1 \cup \mathcal{V}_2, \{f_i \cup g_j | i = 1..m_1, j = 1..m_2\})$

Two important properties of simple hypergraphs derive from the above definitions:

1) $Tr(\mathcal{H}) = Tr(Min(\mathcal{H}))$, that explains why it is customary to represent the input to any transversal-computation algorithm in its minimal form.
2) $Tr(\mathcal{H}_1 \cup \mathcal{H}_2) = Min(Tr(\mathcal{H}_1) \vee Tr(\mathcal{H}_2))$, that serves as the foundation for the most famous transversal-computation strategy, the Berge-multiplication.

Formal proofs for both properties can be found in [8], where they appear as corollaries of the *vertex-coloring lemma*.

Aside from its set-oriented definition, hypergraphs are commonly specified only by their incidence matrix, which is a $m \times n$ matrix $A = [a_{ij}]_{m \times n}$ whose columns and rows correspond to the vertices and edges respectively, in such a way that $a_{ij} = 1$ if $v_i \in \mathcal{E}_j$ and $a_{ij} = 0$ otherwise.

III. TAXONOMY OF TRANSVERSAL-COMPUTATION ALGORITHMS

It can be easily proved that no algorithm can find all the minimal hitting sets in polynomial time with respect to the size of its input [1]. However, there are definitely some algorithms that exhibit a more efficient strategy than others while computing the set of all minimal hitting sets. According to the strategy they follow, transversal-computation algorithms can be classified into two groups: those that follow an incremental test-and-generate strategy, and those that follow a space-delimited search-with-jumps strategy.

A. INCREMENTAL TEST-AND-GENERATE STRATEGY

The first group, algorithms that follow an incremental test-and-generate strategy, is represented by algorithms, most of which were developed within the context of hypergraph theory. The idea of these algorithms is to work with only a subset of edges of the given hypergraph, compute the complete set of minimal hitting sets in that subgraph, and then incrementally add more edges to the analysis and update the accumulated answer set. Commonly, the initial subgraph consists of only one edge and the algorithm's main loop analyzes one more edge with each iteration, updating the answer set until the whole hypergraph has been analyzed. Evidently, these algorithms do not require the complete hypergraph or its incidence matrix on input, which makes them ideal for modeling dynamic phenomena. Examples of these algorithms include the basic *Berge-multiplication* algorithm [8], Dong and Li's (*DL*) algorithm [9], and Kavvadias and Stavropoulos' (*KS*) algorithm [2]. A variant in this same group, that comes from testor theory, is the *YYC* algorithm [10]. The differences from one algorithm to another lie in the strategies they use to update the set of *MHS*.

One of the earliest approaches in this group is the *Berge-multiplication* algorithm [8]. Apparently, this algorithm dates back much further and has been rediscovered several times for various applications [2]. Nevertheless,

the Berge-multiplication algorithm generates the transversal hypergraph $Tr(\mathcal{H})$ (i.e. finds the complete family of minimal hitting sets) of a given hypergraph $\mathcal{H}$, by analyzing its input one edge at a time, and generating a partial answer that contains all minimal hitting sets for the *partial hypergraph* conformed by the set of edges already analyzed.

Several algorithms in this group follow the same incremental principle, although they add a number of performance enhancements; most notably Dong & Li's (*DL*)-algorithm [9] which simplifies the minimization process by identifying, on each iteration, the minimal hitting sets that intersect the current analyzed edge. Since there is no need to combine those hitting sets any further, they are just immediately added to the next partial solution set. Also, Kavvadias & Stavropoulos' (*KS*)-algorithm [2] introduces three modifications to the basic Berge-multiplication algorithm: first, the computation of transversals is performed in a depth-first fashion, so partial results can be output on each level. Second, in order to avoid regenerating vertex combinations already examined, and thus prune some branches of the search tree, they use the concept of the *appropriate set*. And finally, the concept of *generalized node* is used to further accelerate the computation.

B. SPACE-DELIMITED SEARCH-WITH-JUMPS STRATEGY

The second group, algorithms that use a space-delimited search-with-jumps strategy, is somewhat bigger and can be in turn divided into two sub-groups: algorithms that follow a divide-and-conquer approach (i.e. search-space delimitation), and algorithms that follow an open search-with-jumps strategy.

The idea of using a divide-and-conquer approach comes initially from the domain of Boolean algebras. As the label suggests, the algorithms in this group look to subdivide their initial input (set, hypergraph or matrix) into smaller chunks, compute the *MHS* in a small search-space and then combine the results yield by each search. This process can be done recursively until the entries are small enough to apply a trivial algorithm. The difference between algorithms within this sub-group lies fundamentally in the way in which the partition of the original entry is performed, as well as in the way partial results are combined. Probably, the most representative algorithm using this approach is Fredman and Khachiyan's (*FK*) algorithm [11]. However, latest proposals include the parallelization of the search process as in [12] or the use of binary decision diagrams [13] as proposed by [14].

From testor theory comes a group of algorithms that perform an ordered search for *MHS* over the power set of vertices. Following some predefined search order, each subset of vertices is tested to determine if it is an *MHS* or not. However, the search process is not exhaustive. Some properties of each tested subset allow the algorithm to infer which other successive subsets, following the established order, cannot possibly be *MHS*, and consequently decide that those subsets are not worth to be tested. The act of bypassing the test of some subsets is commonly referred to as jumping. In general, the selected order for traversing the power set of vertices, along with the magnitude of the jumps (i.e. the number of subsets not tested), and the specific procedure applied to a subset for testing if it is a *MHS* or not, determine the differences among algorithms of this last sub-group. Representative algorithms of this strategy are *LEX* [15], *FastCT* [16], and most of all, the *Binary-Recursive* (*BR*) algorithm [17]) that orders the edges in the input incidence matrix by increasing cardinality, and then searches the space of vertex subsets following a particular order which is a combination of the cardinality and lexicographic orders. It also includes the same *appropriate* concept used in the *KS* algorithm (known as *compatible set*) and combines hitting sets from different levels of the ordered search tree.

C. MIXED STRATEGY

There is also a small number of algorithms that follow a mixed strategy, that is, they incorporate some elements and ideas from the incremental test-and-generate group, and some other elements from the space-delimited search group. This kind of algorithms will not be analyzed further in this paper, not only because of its small number of elements, but because of the fact that, since the proposed learning strategy can be applied to both latter groups, it can also be applied to the mixed strategy group of algorithms. Probably the most representative algorithm of this mixed strategy group is the *Bailey-Manoukian-Ramamohanarao* (*BMR*)-algorithm [18] which first orders the set of vertices by increasing number of hits on the edges, then it recursively partitions the set of edges of the input hypergraph, by iterating over the vertices and testing the partition induced by each vertex. When a partition is sufficiently small, the DL-algorithm is invoked as a subroutine (or any other algorithm from the Brege-mutiplication's family) to find the set of minimal hitting sets.

IV. THE PROPOSED LEARNING STRATEGY

Any transversal-computation algorithm, regardless of the taxonomic branch it belongs to, selects an initial vertex subset to test and determines if the selected subset is a hitting-set and if it is minimal. Upon deciding on that matter, the algorithm updates its accumulated answer accordingly, and uses its own strategy to specify the next vertex subset to be tested. All known search strategies are based on verifying the conditions needed by a vertex subset to be hitting and to be minimal (See Definition 2). However, there are many more conditions that lead to vertex subsets that are neither hitting nor minimal. Therefore, finding the opposite of what the algorithm is searching for (e.g. vertex subsets that are neither hitting nor minimal) can help to reduce the search space in a particular problem.

The proposed learning strategy takes advantage of general background knowledge about the search space and uses the local knowledge that the host algorithm learns (e.g. if a particular vertex subset is hitting and minimal) in order to identify *incompatible* vertex combinations that cannot possibly be part of any minimal hitting set. The learning strategy puts

that knowledge at the disposal of the algorithm, allowing it to avoid (mask or jump) the test of any vertex subset that contains such combinations.

The identification of incompatible vertex combinations is performed at a structural level, inside the incidence matrix of the input hypergraph, using the following concept:

Definition 5 (Restricted Matrix): Let $\mathcal{A}_{m\times n}$ be the incidence matrix of a simple hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, and let $\tau \subseteq \mathcal{V}$. Then, the *$\tau$-restricted matrix* $\mathcal{A}|_\tau$ is the submatrix obtained by removing from $\mathcal{A}$ all columns (vertices) that are not in τ.

Using such restricted matrix, it is possible to independently characterize hitting sets and minimal sets in structural (symbolic) terms as follows:

Definition 6 (Structural Characterization of a Hitting Set): A subset $\tau \subseteq \mathcal{V}$ is a *hitting set* in hypergraph $\mathcal{H}$ iff the submatrix $\mathcal{A}|_\tau$ does not contain any row composed exclusively by zeros (a zero-row).

Definition 7 (Structural Characterization of a Minimal Set): A subset $\tau \subseteq \mathcal{V}$ is *minimal* iff $A|_\tau$ contains all the rows of an identity matrix I_n where $n = |\tau|$.

Evidently, if a subset of vertices is a hitting set and it is also minimal, then it is a minimal hitting set. Therefore, besides searching for vertex subsets that satisfy Definition 2, it is possible to search for vertex subsets that satisfy both definitions 6 and 7. The apparently subtle change from Definition 2 to Definitions 6 and 7 turns out to be crucial because it allows to make a complete list of all the possible property combinations that an arbitrary transversal-computation algorithm may find during its search. As a consecuence, the search space can be clearly partitioned and a specific search strategy can be defined for each search space class.

This structural characterization of the minimal hitting sets puts in evidence a particular relation between some vertices:

Definition 8 (Domination): (*Single domination*) Vertex v_1 *dominates* vertex v_2 iff $(\forall \mathcal{E}_i \in \mathcal{E})[v_2 \in \mathcal{E}_i \Rightarrow v_1 \in \mathcal{E}_i]$, that is, if in any row of the incidence matrix $\mathcal{A}$ where v_2 has a value 1, v_1 also has it.

This phenomenon can also be found as a subset of vertices dominating another vertex (*Multiple domination*). In any case, when there exists a domination between any two vertices or between a subset and a vertex, we say that those elements are *incompatible*, and therefore cannot both be part of any minimal hitting set.

From the above definitions, it follows that the search space for any transversal-finding problem can always be partitioned into four classes:

- Hitting sets (not minimal): those subsets that satisfy Definition 6, but not Definition 7.
- Minimal sets (not hitting): satisfying Definition 7, but not Definition 6.
- Minimal hitting sets: satisfying both definitions 6 and 7.
- Undetermined sets: satisfying neither Definition 6 nor Definition 7.

The logic for the learning strategy relies on the following background knowledge expressed as rules:

1) Finding a hitting set implies that all its supersets must be excluded from the search process since they cannot be minimal.
2) Finding a minimal set implies that all its subsets must be excluded from the search since they cannot be hitting.
3) Finding a minimal hitting set implies that all its subsets and supersets must be excluded from further search.
4) Any other subset found (undetermined) always contains, at least one pair of incompatible vertices and must also be excluded from further search.

While the knowledge expressed by these rules may seem to be obvious, it becomes crucial to understand that no known tranversal-computation algorithm implements the full set of rules. Since minimal hitting sets are at the same time hitting and minimal, some algorithms traverse the family of hitting sets, searching for subsets of those with the potential to be minimal. Other algorithms traverse the family of minimal sets and search for candidate vertices to complete a hitting set. By partitioning the search space into four clearly defined classes and defining a complete set of rules, the learning strategy is guaranteed to complement the search strategy for any transversal-computation algorithm.

The main goal at designing the learning strategy was to preserve the search nature of the host transversal-computation algorithm while providing it with useful information that can potentially enhance its search process by showing which subsets are not worth testing.

V. IMPLEMENTING THE STRATEGY

For implementation purposes it must be noted that rules 1, 2 and 3 above, can only yield a *tabu-list* [19] as their result (e.g. a list of items categorically excluded from further test). Generating such a list has two severe drawbacks. First, since the list would have to be exhaustively checked to decide if a particular subset is to be excluded from the search, even a slight grow in size would severely ruin the host algorithm's performance. Second, the knowledge contained in rules 1 to 3 is precisely what all transversal-computation algorithms embed into their search strategies, therefore its implementation would not provide any useful information to the host algorithm.

On the other hand, rule 4 expresses knowledge that is neither commonly used by transversal-computation algorithms, nor a potential risk to the algorithm's performance. Vertex incompatibilities can be learned as the host algorithm searches for hitting or minimal sets, and the use of an optimized suitable representation of that knowledge minimizes the risk of yielding information already known by the algorithm, and reduces the overhead of deciding how to proceed with a particular vertex subset. The learned incompatibilities can therefore allow the algorithm to avoid unnecessary testing, thus increasing its efficiency.

Following this line of reasoning, the learning strategy is implemented as a procedure *UpdateIncompatibilities*() which creates and updates a knowledge table containing the

The *UpdateIncompatibilities*() procedure
Input: A vertex subset $\tau = \{x_1, ..., x_r\}$
Local data: The table K of accumulated knowledge
Effect: Updates K with the incompatibilities inferred from τ

For each $x_i \in \tau$ **do**
  $\boldsymbol{D}$ = *Dominators*(x_i)
  If $\boldsymbol{D} \neq \emptyset$ **then**
    For all $d \in \boldsymbol{D}$ **do**
      If $d \cap \tau = \emptyset$ **then**
        RemoveDominator(x_i, d)
        For each($x_j \in \tau \backslash x_i$) **do**
          Unless $(\exists s \in \boldsymbol{D})[s \subseteq d \cup \{x_j\}]$
            AddDominator(x_i, $d \cup \{x_j\}$)
  Else *;when x_i has no registered dominators...*
    For each $x_j \in \tau \backslash x_i$ **do**
      AddDominator(x_i, x_j)

FIGURE 1. Pseudocode for the ***UpdateIncompatibilities*() procedure.**

discovered incompatibilities. This *learned knowledge* is combined with the *background knowledge* about the four classes in which the search space is always partitioned. The host algorithm calls *UpdateIncompatibilities*() each time it selects a vertex subset to test.

The interaction between the host algorithm and the learning strategy proceeds as follows:

- Depending on the taxonomical branch of the host algorithm, it uses its own predefined search order for selecting a vertex subset to test.
- If the tested subset is not a minimal hitting set, then the host algorithm calls *UpdateIncompatibilities*() with the selected subset as argument (See Figure 1.)
- At any moment, the host algorithm can query the accumulated knowledge expressed as an incompatibility catalog in order to decide about future vertex subset selections for testing.

The reason why the learning strategy is labeled as symbolic is the optimized representation of the incompatibilities learned. Both, single and multiple incompatibilities learned, are represented by a unique domination operator. The semantics of a domination, combined with the background knowledge in definitions 6 and 7, provide all the foundation needed to define the logic for updating the incompatibilities table.

Pseudocode for procedure *UpdateIncompatibilities*() is shown in Figure 1. It makes use of three auxiliary functions whose semantics are described as follows:
AddDominator(x,y) and *RemoveDominator(x,y)* allow the edition of the incompatibilities table, and *Dominators(x)* returns a set containing all vertex or vertex subsets that dominate vertex *x*.

VI. EXPERIMENTAL RESULTS

As it was stated before, the goal of the proposed learning strategy is to enhance the performance of searching algorithms that look for minimal hitting sets. In order to assess the change in efficiency resulting from adding the learning strategy to a transversal-computation algorithm, we selected two host algorithms, one from each group described in Section 3.

The performance of a general search algorithm can be assessed in several different ways. For the particular case study, two criteria are most relevant: the number of vertex subsets the algorithm tests, and of course, the global execution time for solving any problem instance. In an hypothetical ideal case, a *perfect* algorithm, would only test those elements in the search space that are correct solutions to its search. That is, an ideal traversal-computation algorithm would not *search* but only pick all the correct answers from its search space. Since clearly no ideal algorithm exists, the ratio between the total number of answers in the search space, and the number of elements that the algorithm actually tests (which we have labeled as *tests*) is proposed as an adequate measure for the performance enhancement. Consequently, we choose to measure the *efficiency* of a transversal generation algorithm $\mathcal{A}(\mathcal{H})$ as the ratio between the total number of minimal hitting sets in $\mathcal{H}$, and the number of tests performed by the algorithm.

From the incremental test-and-generate strategy group we selected the *YYC*-algorithm [10], a slight variant of the Berge-multiplication that includes the identification of sets that do not need to be combined (as in the *DL*-algorithm), a reordering of the analysis of edges (similar as in the *BMR*-algorithm), and the concept of appropriate set to select the needed combinations (as in the *KS*-algorithm). From the space-delimited search-with-jumps strategy group we selected the *BM*-algorithm [17] as described in Section 3. Both algorithms originated within the testor theory field, and they were selected for their representativity in terms of techniques used, as well as for their recent appearance in indexed journal articles [20], [21].

A. FOUNDATION AND DESCRIPTION OF PERFORMED EXPERIMENTS

Acknowledging the presence of a *no-free-lunch* effect, authors recognize the importance of avoiding the bad habit of testing an algorithm with a limited and biased set of problems. When such bad habit is not avoided, the generality and credibility of experimental results is generally questioned. In order to test the change in performance derived from adding the proposed learning strategy to any transversal-computation algorithm it is critical to ensure that the set of test problems is not biased and that it covers a wide range of possible input models. In [22] a set of matrix operators is proposed that allows—starting from very small incidence matrices whose transversal is known—to construct large matrices whose transversal can be computed using a single formula (i.e. without the need for a transversal-computing algorithm). When these operators are systematically used, they allow the construction of a benchmark test set that includes the three most relevant types of hypergraphs, those whose set of vertices grows exponentially, those whose set of edges grow exponentially, and those with both—the sets of vertices and

the set of edges grow simultaneously. These three scenarios have been shown to cover a great majority[1] of computational behaviors in transversal-computation algorithms [23]. Therefore, during this experiments the methodological guide stated in [23] is strictly followed, so ensuring the generality of the obtained results for a wide range of transversal-computation algorithms.

B. TERMINOLOGY AND LABELING

On each scenario the efficiency and run time of the original algorithm are compared with those of the adapted version (labeled as *YYC** and *BR**).

The *YYC* algorithm is generally considered to be more efficient than the *BR* algorithm. However, because of its incremental nature, *YYC* is better suited for working with hypergraphs with a moderately small number of edges but a high number of vertices. On the contrary, the *BR* algorithm, because of its vertex-oriented search approach, performs better with a small number of vertices. On scenario #1, we tested all four algorithms (the original versions and the adapted versions) with hypergraphs with a linearly increasing number of vertices. As expected, the addition of the learning strategy slightly increases the run time of all four algorithms, although only for small hypergraphs.

On the graphics, as well as for all subsequent scenarios, *BR* and *BR** lines are gray color and have a solid fill circular marker, while *YYC* and *YYC** lines are black and display a cross marker. See Figures 2 and 3. Results for *YYC* and *BR* algorithms are shown in Tables 1 and 2, respectively.

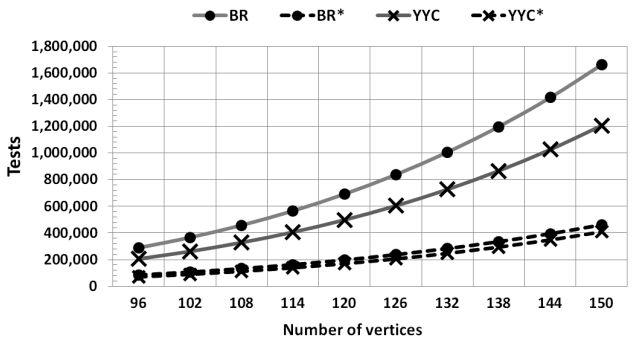

FIGURE 2. Scenario #1 comparative results for number of tests.

On scenario #2, we again tested all four algorithms, but now against hypergraphs with an exponentially increasing number of edges. As was expected both algorithms performed efficiently for small number of edges, but as the number increases the difference starts to show, as well as the gain resulting from using the learning strategy. In this scenario, since the tests and run time scales are notoriously different, we provide separate graphics to facilitate the results interpretation. See Figures 4, 5, 6, and 7. Results for YYC and BR algorithms are shown in Tables 3 and 4, respectively.

[1]With the notable exception of a small family of *stair-like* incidence matrices. However, there are no known transversal-computational algorithms whose best or worse performance cases occur solving hypergraphs with this type of incidence matrices.

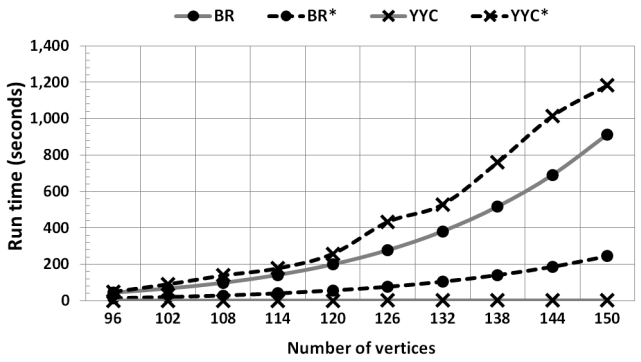

FIGURE 3. Scenario #1 comparative results for run time.

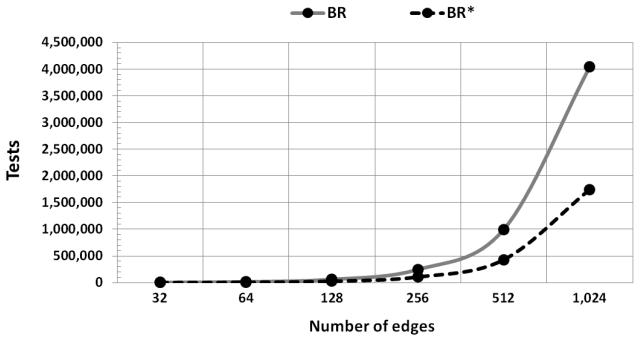

FIGURE 4. Scenario #2 Number of tests for the BR and BR* algorithms.

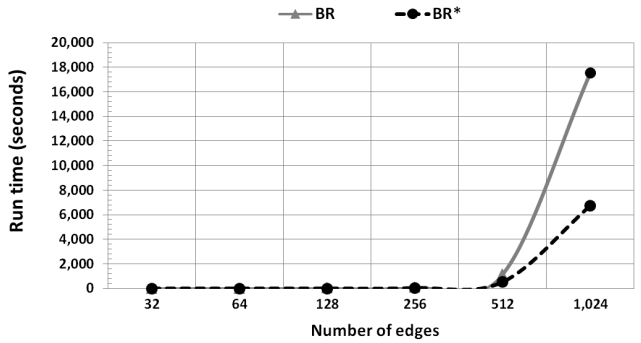

FIGURE 5. Scenario #2 Run time of the BR and BR* algorithms.

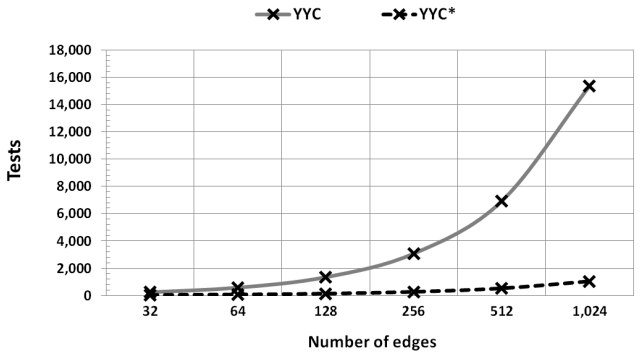

FIGURE 6. Scenario #2 Number of tests for the YYC and YYC* algorithms.

TABLE 1. YYC tests, Scenario #1.

edges	vertices	minimal hitting sets	YYC tests	YYC* tests	YYC time (sec)	YYC* time (sec)
4	96	66,304	206,112	70,688	0.14	48.48
4	102	84,388	261,868	89,624	0.21	89.73
4	108	105,948	328,248	112,140	0.24	138.044
4	114	131,404	406,524	138,662	0.30	177.22
4	120	161,200	498,040	169,640	0.38	257.34
4	126	195,804	604,212	205,548	0.45	432.33
4	132	235,708	726,528	246,884	0.55	527.98
4	138	281,428	866,548	294,170	0.67	759.53
4	144	333,504	1,025,904	347,952	0.79	1,014.34
4	150	392,500	1,206,300	408,800	0.97	1,182.07

TABLE 2. BR tests, Scenario #1.

edges	vertices	minimal hitting sets	BR tests	BR* tests	BR time (sec)	BR* time (sec)
4	96	66,304	289,296	85,248	44.50	13.60
4	102	84,388	366,469	106,930	66.81	19.94
4	108	105,948	458,154	132,516	98.06	28.06
4	114	131,404	566,067	162,450	141.26	39.83
4	120	161,200	692,020	197,200	199.52	56.06
4	126	195,804	837,921	237,258	277.66	76.19
4	132	235,708	1,005,774	283,140	381.17	104.22
4	138	281,428	1,197,679	335,386	516.95	140.24
4	144	333,504	1,415,832	394,560	690.97	185.94
4	150	392,500	1,662,525	461,250	913.69	244.21

TABLE 3. YYC tests, Scenario #2.

edges	vertices	minimal hitting sets	YYC tests	YYC* tests	YYC time (sec)	YYC* time (sec)
32	20	10	240	41	0.0006	0.0012
64	24	12	576	75	0.0027	0.0040
128	28	14	1,344	141	0.0119	0.0253
256	32	16	3,072	271	0.0599	0.0397
512	36	18	6,912	529	0.2597	0.1155
1,024	40	20	15,360	1,043	0.2005	0.3259

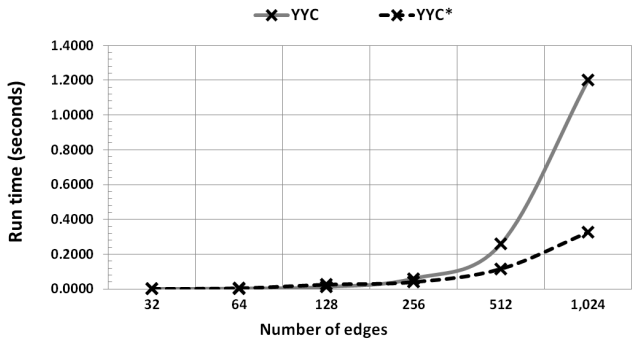

FIGURE 7. Scenario #2 Run time of the YYC and YYC* algorithms.

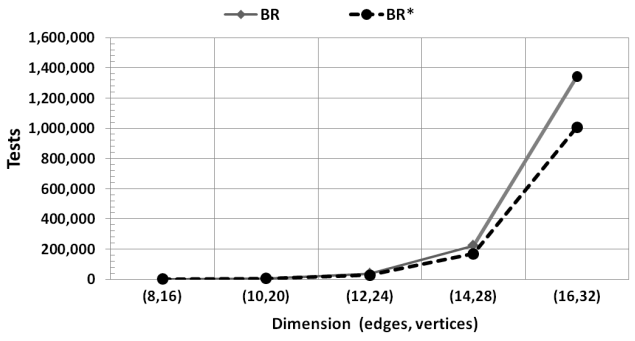

FIGURE 8. Scenario #3 comparative results for number of tests.

Finally, in scenario #3 we test with proportionally bigger hypergraphs, both in number of edges and vertices. This last scenario reveals the best case for the *YYC* algorithm, showing that these exact type of hypergraphs are the ones where the original version of the *YYC* algorithm naturally learns all that there is to learn. Thus, the addition of the learning strategy

TABLE 4. BR tests, Scenario #2.

edges	vertices	minimal hitting sets	BR tests	BR* tests	BR time (sec)	BR* time (sec)
32	20	10	3,394	1,587	0.04309	0.03275
64	24	12	14, 371	6,339	0.52801	0.30987
128	28	14	59,686	25,881	6.52980	3.58238
256	32	16	244,975	105,455	82.9799	40.6052
512	36	18	997, 834	430,247	1,153.36	528.429
1,024	40	20	4,043,611	1,747,211	17,556.93	6,742.74

TABLE 5. YYC tests, Scenario #3.

edges	vertices	minimal hitting sets	YYC tests	YYC* tests	YYC time (sec)	YYC* time (sec)
8	16	16	45	45	0.00015	0.00004
10	20	32	93	93	0.00039	0.00004
12	24	64	189	189	0.00110	0.00006
14	28	128	381	381	0.00299	0.00010
16	32	256	765	765	0.00769	0.00020

TABLE 6. BR tests, Scenario #3.

edges	vertices	minimal hitting sets	BR tests	BR* tests	BR time (sec)	BR* time (sec)
8	16	16	1,038	782	0.01089	0.00712
10	20	32	6,222	4,671	0.30779	0.21321
12	24	64	37,326	28,000	8.49921	5.94410
14	28	128	223,950	167,969	291.716	193.976
16	32	256	1,343,694	1,007,778	10,721.02	6,796.02

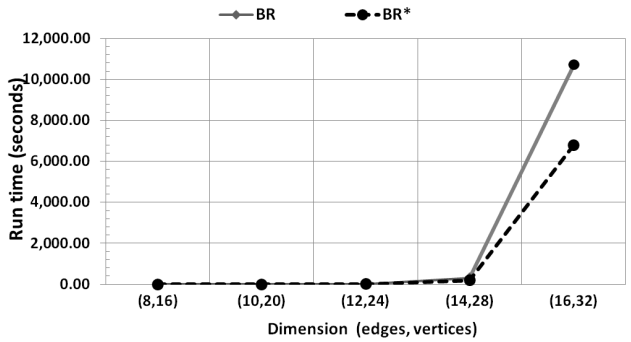

FIGURE 9. Scenario #3 comparative results for run time.

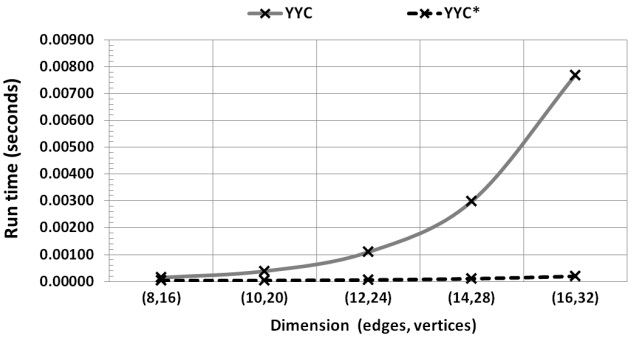

FIGURE 11. Scenario #3 comparative results for run time.

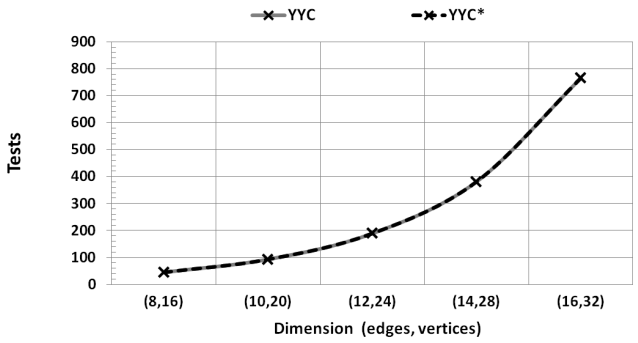

FIGURE 10. Scenario #3 comparative results for number of tests.

causes no effect in the number of tests but a great efficiency increase in run time. See Figures 8, 9, 10, and 11. Results for *YYC* and *BR* algorithms are shown in Tables 5 and 6, respectively.

VII. CONCLUSIONS AND FUTURE RESEARCH

We have presented a general symbolic learning strategy for guiding the search process in transversal-computation algorithms. The goal of the strategy is to supply the host algorithm with extra knowledge about the problem search space, as well as to enrich its local knowledge about the traversal process, thus enhancing its general efficiency measured both as the number of tests performed and the run-time achieved. At the symbolic level the strategy relies on finding vertex incompatibilities.

The contribution of this research can be neatly summarized by two elements: First, the independent characterization of hitting and minimal sets allowing the search space for any transversal-computation algorithm to be partitioned into four classes and the definition of the background rules implied by that partition. Second, the concrete algorithm for learning

single and multiple vertex dominations (called *incompatibilities*) as an efficient way of implementing a multi-algorithm learning strategy. The intended general contribution is to show how symbolic learning algorithms can open a partially unexplored new route for optimizing solutions for combinatorial problems.

Although a *no-free-lunch* effect for this kind of algorithms has always been known, on very rare occasion an algorithm or family of related algorithms is studied in detail to characterize its computational behavior against different models of input. Our experimental results have shown that the proposed learning strategy is able to enhance the efficiency of algorithms in any of the two identified groups, albeit the gain is only worth for large hypergraphs. If the learning strategy is activated for a problem with a small input hypergraph, it will enhance the tests performance, but at a cost in running time. The real benefit of the proposed learning strategy becomes evident as the size of the input hypergraph increases. Moreover, the proposed strategy revealed itself as particularly useful when dealing with hypergraphs with a large number of vertices or edges, but particularly when the host algorithm does not follow a Berge-multiplication strategy. The reason for this effect seems to be the larger amount of incompatibilities learned by the strategy, in combination with poor detection and filtering mechanisms of the host algorithm.

The characteristic behavior of the experimented algorithms and the global relevance of the TransHyp problem, combined with its known non-polynomial complexity, points out that a feasible next step in research can be the design of a performance-directed expert system able to recommend the best algorithm to solve a specific input hypergraph, or at least warn about algorithms believed to have the poorest performance with that particular input. Meanwhile, a detailed study about different models in which the learning strategy can be parallelized would not only increase its performance and soothe its integration with different host algorithms, but also pave the way to the design of such expert system.

REFERENCES

[1] K. Murakami and T. Uno, "Efficient algorithms for dualizing large-scale hypergraphs," *Discrete Appl. Math.*, vol. 170, pp. 83–94, Jun. 2014.
[2] D. J. Kavvadias and E. C. Stavropoulos, "An efficient algorithm for the transversal hypergraph generation," *J. Graph Algorithms Appl.*, vol. 9, no. 2, pp. 239–264, 2005.
[3] J. R. Slagle, C.-L. Chang, and R. C. T. Lee, "A new algorithm for generating prime implicants," *IEEE Trans. Comput.*, vol. C-100, no. 4, pp. 304–310, Apr. 1970.
[4] M. Hagen, *Algorithmic and Computational Complexity Issues of MONET*. Göttingen, Germany: Cuvillier Verlag, 2008.
[5] M. Hagen, "Lower bounds for three algorithms for transversal hypergraph generation," *Discrete Appl. Math.*, vol. 157, no. 7, pp. 1460–1469, 2009.
[6] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*. Springer, 2013.
[7] V. I. González-Guevara, S. Godoy-Calderon, E. Alba-Cabrera, and J. Ibarra-Fiallo, "A mixed learning strategy for finding typical testors in large datasets," in *Proc. Iberoamerican Congr. Pattern Recognit.* Montevideo, Uruguay: Springer, 2015, pp. 716–723.
[8] C. Berge, *Hypergraphs: Combinatorics of Finite Sets*. Amsterdam, The Netherlands: Elsevier, 1984, vol. 45.
[9] G. Dong and J. Li, "Mining border descriptions of emerging patterns from dataset pairs," *Knowl. Inf. Syst.*, vol. 8, no. 2, pp. 178–202, 2005.
[10] E. Alba-Cabrera, J. Ibarra-Fiallo, S. Godoy-Calderon, and F. Cervantes-Alonso, "YYC: A fast performance incremental algorithm for finding typical testors," in *Proc. Iberoamerican Congr. Pattern Recognit.* Puerto Vallarta, Mexico: Springer, 2014, pp. 416–423.
[11] M. L. Fredman and L. Khachiyan, "On the complexity of dualization of monotone disjunctive normal forms," *J. Algorithms*, vol. 21, no. 3, pp. 618–628, 1996.
[12] C. E. Leiserson, M. Moreno Maza, L. Li, and Y. Xie, "Parallel computation of the minimal elements of a poset," in *Proc. 4th Int. Workshop Parallel Symbolic Comput.*, 2010, pp. 53–62.
[13] D. E. Knuth, *The Art of Computer Programming: Combinatorial Algorithms, Part 1*, vol. 4A. New Delhi, India: Pearson Education, 2011.
[14] T. Toda, "Hypergraph transversal computation with binary decision diagrams," in *Proc. Int. Symp. Exp. Algorithms*. Rome, Italy: Springer, 2013, pp. 91–102.
[15] Y. Santiesteban-Alganza and A. Pons-Porrata, "LEX: A new algorithm for calculating typical testors," (in Spanish), *Revista Ciencias Matemáticas*, vol. 21, no. 1, pp. 85–95, 2003.
[16] G. Sanchez-Diaz, M. Lazo-Cortes, and I. Piza-Davila, "A fast implementation for the typical testor property identification based on an accumulative binary tuple," *Int. J. Comput. Intell. Syst.*, vol. 5, no. 6, pp. 1025–1039, 2012.
[17] A. Lias-Rodríguez and A. Pons-Porrata, "BR: A new method for computing all typical testors," *Prog. Pattern Recognit., Image Anal., Comput. Vis., Appl.*, 2009, pp. 433–440.
[18] J. Bailey, T. Manoukian, and K. Ramamohanarao, "A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns," in *Proc. ICDM*, vol. 3, 2003, p. 485.
[19] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.
[20] V. Rodríguez-Diez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, and M. S. Lazo-Cortés, "Fast-BR vs. fast-CT_EXT: An empirical performance study," in *Proc. Mexican Conf. Pattern Recognit.* Huatulco, Mexico: Springer, 2017, pp. 127–136.
[21] I. Piza-Davila, G. Sanchez-Diaz, M. S. Lazo-Cortes, and C. Noyola-Medrano, "Enhancing the performance of yyc algorithm useful to generate irreducible testors," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 32, no. 1, p. 1860001, 2017.
[22] E. Alba-Cabrera, J. Ibarra-Fiallo, and S. Godoy-Calderon, "A theoretical and practical framework for assessing the computational behavior of typical testor-finding algorithms," in *Proc. Iberoamerican Congr. Pattern Recognit.* Havana, Cuba: Springer, 2013, pp. 351–358.
[23] E. Alba-Cabrera, S. Godoy-Calderon, and J. Ibarra-Fiallo, "Generating synthetic test matrices as a benchmark for the computational behavior of typical testor-finding algorithms," *Pattern Recognit. Lett.*, vol. 80, pp. 46–51, 2016.

VÍCTOR IVÁN GONZÁLEZ-GUEVARA received the M.Sc. degree from the Centro de Investigación en Computación, Instituto Politécnico Nacional. He is currently developing a SaaS platform for the BBVA Group (financial services with artificial intelligence). His research interests include the classic algorithms of typical testors, pattern recognition, rule-based microworld generation systems, and artificial intelligence.

SALVADOR GODOY-CALDERON was born in Mexico City, Mexico, in 1968. He graduated in computer engineering from ITAM, Mexico, in 1992. He received the M.Sc. degree from CINVESTAV, in 1994, and the Ph.D. degree in computer sciences from the Center for Computing Research, Instituto Politécnico Nacional, Mexico, in 2006, where he is currently the Head of the Artificial Intelligence Laboratory. His research interests include pattern recognition, testor theory, and logic and symbolic systems.

EDUARDO ALBA-CABRERA received the M.Sc. degree in mathematics from Kharkiv National University, Ukraine, and the Ph.D. degree in mathematics from the Instituto de Cibernética, Matemática y Física de La Habana, Cuba. Since 2001, he has been a Professor and a Researcher with the Universidad San Francisco de Quito, Ecuador, where he has been an Associated Dean of the Science and Engineering School, since 2015. His main research interests include pattern recognition, discrete algorithms, and testor theory.

HIRAM CALVO received the Ph.D. degree (Hons.) in computer science from the Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), Mexico, in 2006. His Ph.D. thesis was on the Spanish syntax analyzer DILUCT. Since 2006, he has been a Lecturer with CIC-IPN. He did a Postdoctoral Stay at the Nara Institute of Science and Technology, Japan, from 2008 to 2010. He is currently a full-time Research Professor at CIC-IPN, and a member of the National Research System (CONACYT-SNI) level II. His research interests include lexical semantics, pattern recognition, similarity measures, and author profiling. He was awarded with the Lázaro Cárdenas Prize by the President of Mexico, in 2006.