Jose Torres-Jimenez[1], David Romero[2], Eduardo Rodriguez-Tello[1] and Federico Zertuche[2]
[1]Cinvestav-Tamaulipas, Cd. Victoria, Tam., México
{jtj, ertello}@cinvestav.mx
[2]Instituto de Matemáticas, UNAM, Cuernavaca, Mor., México
{davidr, zertuche}@matcuer.unam.mx

A Hypergraph Algebra for Generating SAT Instances

Abstract

The satisfiability (SAT) problem is central in computational complexity theory. The SAT problem consists in finding a truth assignment for the variables in a formula stated in *conjunctive normal form*. Its relevance comes both from the fact that it was the first member of the NP-complete class, and many important combinatorial problems might be solved using it as a bridge (i.e., by transforming the original problem to SAT, solving the SAT instance and obtaining the result for the original problem using the solution for the SAT instance). Therefore, it is interesting to develop effective exact or approximate solution algorithms. The effectiveness of a solution algorithm can be measured using either well known benchmarks or SAT instances randomly generated. In this paper we present a hypergraph algebra for generating random SAT instances. This algebra has many advantages with respect to other SAT generators, it allows to produce instances with controlled relations of variables, distribution of signs or variables, etc.
Keywords: SAT problem, random SAT instance, hypergraphs.

1 Introduction

Given a logic formula in conjunctive normal form (CNF), the *satisfiability (SAT) problem* is to determine the values for the variables in such a way that the formula is true (or prove that there is no solution). More precisely, the SAT problem refers to the propositional logic problem that consists in determining the truth values for a set of variables X in a set of clauses C, expressed in CNF. The CNF is $F = C_1 \wedge C_2 \wedge ... \wedge C_n$, where every C_i denotes a clause, which in turn is a disjunction of literals over the set of variables V (a literal is a variable in negated or non-negated form). The set of truth values must evaluate to true the whole formula F.

The SAT problem was the first member of the NP-complete class [1]. Whereas in the theoretical sense SAT is a core problem in mathematical logic and computing theory, in practice it is fundamental to tackle many problems related to automated reasoning, computer aided design, database, and robotics [3]. Furthermore, the so-called 3-SAT problem (each clause consists exactly of three literals) is the most simple and more used problem for testing NP-completeness in computational complexity analysis [2]. As many important combinatorial optimization problems are NP-complete, they can be transformed efficiently to a SAT instance, then solve the resulting SAT instance, and finally return the solution to the original problem [2].

Usually the evaluation of algorithms for solving SAT problem instances is done through the use of well known benchmarks that include: random SAT problem instances, SAT problem instances taken from practical problems, and SAT problem instances obtained using specific models [5]. Another approach to test SAT solving algorithms is the use of a SAT instances random generator. In this paper we propose a hypergraph algebra that enables the construction of a SAT instances random generator with the following characteristics: (a) it has the potential of generating any SAT instance, (b) it produces SAT instances in which all the variables are related, and (c) the distribution of signs and variables is controlled. In addition the new hypergraph algebra allows the formal specification of SAT instances generators.

The rest of the paper is organized as follows: the modeling of SAT instances with hypergraphs is presented in Section 2; then, in Section 3, we describe the proposed hypergraph algebra; Section 4 provides various programs that generate different structures which can be converted into SAT instances; finally, in Section 5 we give conclusions and suggest future work.

2 Representing SAT instances with hypergraphs

A hypergraph $H = (V, E)$ consists of a non empty set $V = \{i = 1, \ldots, n\}$, whose elements are called nodes or vertices, and a family $E = \{e_1, \ldots, e_m\}$ of non empty subsets of V; the elements of E are called hyperedges. If all hyperedges of H have size ≤ 2, then H is simply called *graph* and the elements of E are called *edges*.

A hypergraph is k-uniform if each of its edges is of size k. Figure 1(a) shows an example of a 3-uniform hypergraph, i.e., $V = \{1, 2, 3, 4\}$ and $E = \{\{1, 2, 3\}, \{2, 3, 4\}\}$.

Given that a SAT instance $\mathcal{I}$ consists of a set of clauses where each clause represents a disjunction of literals, $\mathcal{I}$ (over the set of variables V) can be represented by a hypergraph $H = (V, E)$ as follows. The set V corresponds to the set of variables in $\mathcal{I}$, and E corresponds to the set of clauses in $\mathcal{I}$, i.e., for each clause there is an hyperedge in E containing the nodes that correspond to the variables in the clause. In this way a k-SAT instance is represented by a k-hypergraph in which the clauses are k-hyperedges. Figure 1(b) represents clauses that involve the A, B, C, D, and E variables.

The previous representation does not indicate the sign of variables in the clauses. Hence a special notation will be used for the negated variables: a circle joining the hyperedge that connects to the node that represents the negated variable, the designation that will be used for the signed hypergraph is σHypergraph (σH). The Figure 1(c) represents the clauses (A,B,C) and (C,$\sim D$,E).

Then, a SAT instance can contain two or more clauses involving the same variables but with a different sign combination. A σH can have two or more hyperedges containing the same set of nodes but with different signs. Figure 1(d) shows a σH that represents the clauses ($\sim A$,B,$\sim C$) $\wedge$ (A,$\sim B$,C).

3 An algebra for generating SAT instances

In this section an algebra for manipulating σH objects will be defined, a special feature of σH objects is that the labels of the nodes are K-Dimensional.

The main aim of this σH algebra lies in the motivation of generating connected SAT instances (i.e. the associated hypergraph is connected).

A σH object with K-Dimensional labels is defined as $\sigma H = (V, E)$, where: V is the K dimensional labeled vertex set with cardinality $|V| = n$ and E is the hyperedge set with cardinality $|E| = m$. V is defined according to Equation 1.

In Equation 2 the definition of E is presented, $\diamond$ indicates the sign of the variable and Λ indicates no sign.

$$V = \{v_1, \ldots, v_n\} | (v_i = (x_{i,1}, \ldots, x_{i,K})) \wedge (x_{i,j} \in Z) \tag{1}$$

$$E = \{e_1, ..., e_m\} | (e_i = \{\diamond(\nu_{i,1}, \ldots), \ldots\}) \wedge (\nu_{i,j} \in V) \wedge ((\diamond = -) \vee ((\diamond = \Lambda)) \tag{2}$$

Graphically a vertex that appears as negated in a hyperedge is represented with a dot in the vertex-hyperedge joint.

The basic operations of the σH algebra are defined next:

- Create. Specified with an equal sign, creates a σH object, giving the V and E sets. v.gr. for $K = 2$, $S = \{\{\}, \{\}\}$ creates an empty σH object, and the Equation 3 creates a σH object named R, whose graphic representation can be seen in the Figure 1(e).

$$R = \{\{(2,2),(3,4),(5,6),(7,8)\}, \{-(2,2),(3,4),(7,8)\}, \{-(3,4),(5,6),-(7,8)\}\} \tag{3}$$

- Hyperdge Sign Change. This is represented with the minus sign $-$. The operation of this unitary operator is to modify the E set of a σH object, in such a way that a sign is randomly generated for each hyperedge.

- Hyperedge Sign Elimination. This operator is specified with the notation $|alpha|$, an example is: $S = |R|$ (R refers to the Equation 3) creates a σH object, called S, identical to R but without signs in its set E.

- Move. Specified with a set of sub-indices, it creates an isomorphic object taking as basis one previously created σH object, the labels are adjusted according to the sub-indices set (the $i-th$ new index is created adding $i-th$ index to the original one). For example if $T = |R|_{-1,2}$ where R is taken from Equation 3), then T equals to Equation 4. For a graphical representation of T see Figure 1(f).

$$\{\{(1,4),(2,6),(4,8),(6,10)\}, \{(1,4),(2,6),(6,10)\}, \{(2,6),(4,8),(6,10)\}\} \tag{4}$$

- Merge. Specified with $\otimes$, it makes the merging of two σH objects. Assuming that $R =$

(V_R, E_R) and $S = (V_S, E_S)$ are σH objects they can be merged using Equation 5. As can be inferred the Merge operation is commutative, i.e., $R \otimes S = S \otimes R$

$$R \otimes S = \begin{cases} (V_R \cup V_S, E_R \cup E_S) & \text{if} \quad V_R \cap V_S \neq \emptyset \\ \emptyset & \text{if} \quad V_R \cap V_S = \emptyset \end{cases} \tag{5}$$

- Dimension Modification Operator. It is specified as a super-index, which enables the modification of the label's dimensions (K). The conversion of a σH object R into another S with different dimensions is specified using a positive or negative super-index that modifies the labels' dimensions. For example if R is a σH with labels with 4 dimensions an object S with labels with 3 dimensions is creates using the expression $S = R^{-1}$, the vertex set labels for S are computed in a random way guaranteeing that there is no label repetition (as can be inferred this modifies also the vertex set in a consistent way), the signs of the R edge set are conserved in the S object.

Given an expression involving many operators it is necessary to define the following precedence rules:

- The order of evaluation is from left to right.
- The order of operators evaluation is: first move operator (subindex), then dimension updating (upperindex), next hyperedge sign change (unitary minus sign), next absolute operation, and at last merge.
- Use parenthesis to force a desired evaluation order of an expression.

A σH object can have n-plicated hyperedges but with different sign combinations. For example $R \otimes -R \otimes -R$ produces a σH object in which each original hyperedge is repeated three times (but with different sign combination).

4 Algorithms to illustrate the manipulation of σH objects

In this section some examples of algorithms for manipulating σH objects are presented.

In Algorithm 1 it is illustrated how to generate a σH object with 2-Dimensional labels. The result can be observed in Figure 1(g). As it can be seen from the Algorithm 1, very complex σH objects can be created using iteration and/or recursion.

Algorithm 2 depicts how to generate a σH object with unidimensional labels from a σH object with bidimensional labels. A possible result (given that the labels are randomly generated) is shown in Figure 1(h).

In Algorithm 3 it is illustrated how to generate a doubly connected σH object with unidimensional labels through the manipulation of σH objects with bidimensional labels. The resulting S object is illustrated in Figure 1(i), and a possible labeling of the object T is given in Figure 1(j).

```
Algorithm1()
begin
    R = {{(1,1),(2,1),(2,2)},{(1,1),(2,1),(2,2)}};
    S = R;
    for i from 1 to 2 do
        S = S ⊗ R_{i,0};
    end
    return S;
end
```

Algorithm 1: Generation of a σH object with 2-Dimensional labels

```
Algorithm2()
begin
    R = {{(1,1),(2,1),(2,2)},{(1,1),(2,1),(2,2)}};
    S = R;
    for i from 1 to 2 do
        S = S ⊗ R_{i,0};
    end
    T = S^{-1};
    return T;
end
```

Algorithm 2: Generation of a σH with unidimensional labels from a σH with bidimensional labels

5 Algorithms to generate SAT instances from σH

In this section some algorithms to generate SAT instances through the manipulation of σH objects will be presented. The resulting SAT instances have been used in many publications for example in [6], [4], and [7].

The transformation from a σH with unidimensional labels to a SAT instance is very simple, the set of vertices corresponds to the set of variables, and the set of hyperedges corresponds to the set of clauses.

To construct a 3-SAT instance with a structure that we call Simple Triangle we use the Algorithm 4. In the Algorithm 4 n is the number of variables in the instance, Figure 1(k) is the

```
Algorithm3()
begin
    R = {{(1,1),(2,1),(2,2)},{(1,1),(2,1),(2,2)}};
    W = {{(2,2),(2,1),(3,2)},{(2,2),(2,1),(3,2)}};
    S = R ⊗ W;
    for i from 1 to 2 do
        S = S ⊗ R_{i,0} ⊗ W_{i,0};
    end
    T = S^{-1};
    return T;
end
```

Algorithm 3: An algorithm to generate a doubly connected σH object

graphical representation of the S object. Figure 1(l) shows the T object. The resulting SAT instance is the following: $\{(8 \vee 4 \vee \tilde{}1) \wedge (4 \vee \tilde{}6 \vee 7) \wedge (6 \vee 2 \vee \tilde{}3) \wedge (2 \vee 8 \vee \tilde{}5) \wedge (\tilde{}8 \vee 4 \vee 1) \wedge (\tilde{}4 \vee 6 \vee \tilde{}7) \wedge (\tilde{}6 \vee \tilde{}2 \vee 3) \wedge (2 \vee \tilde{}8 \vee \tilde{}5)\}$.

Algorithm4()
begin
  $n = \{n | n \in +N^* \wedge n \; is \; even\}$;
  $R = \{\{(1,1),(2,1),(2,2)\},\{(1,1),(2,1),(2,2)\}\}$;
  $S = R$;
  for i *from* 1 *to* $(n/2) - 2$ **do**
    $S = S \otimes R_{i,0}$;
  end
  $U = \{\{(n/2,1),(1,1),((n/2)+1,2)\},\{(n/2,1),(1,1),((n/2)+1,2)\}\}$;
  $S = -(S \otimes U)$;
  $S = S \otimes -S$;
  $T = S^{-1}$;
  return T;
end

Algorithm 4: Algorithm to generate a Simple Triangle SAT instance

6 Conclusions

In this paper a new hypergraph algebra was proposed. It enables the formalization of SAT instance random generators providing also the means to specify certain important features of a SAT instance, and the classification of the generators looking for similarities in the specification in terms of the hypergraph algebra.

The results of this research redounds in some future lines of work: a) the creation of more powerful operators able to capture some sophisticated features of a SAT instance (for example the clustering of the variables in the clauses); and b) the creation of a compiler able to validate and execute the formal specification in terms of the new hypergraph algebra.

Acknowledgments

This research was partialy funded by the following projects: CONACyT 58554 Cálculo de Covering Arrays, 51623 Fondo Mixto CONACyT y Gobierno del Estado de Tamaulipas, CONACyT 74521 Repatriación.

References

[1] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, 1971.

[2] M. Garey and D. Johnson, editors. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[3] J. Gu, P. Purdom, J. Franco, and B. Wah. Algorithms for the satisfiability problem: a survey. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science American Mathematical Society*, 35(1):19–151, 1997.

[4] E. Rodriguez-Tello and J. Torres-Jimenez. Era: An algorithm for reducing the epistasis of sat problems. In *Springer LNCS Vol. 2724*, pages 1283–1294, 2001.

[5] SATLIB. Satlib - the satisfiability library. *http://www.satlib.org/*, 2005.

[6] J. Segura-Salazar and J. Torres-Jimenez. Hypersat a new generator for 3-sat instances. In *Proceedings of the ICCIMA 2001, IEEEComputer Society ISBN 0-7695-1312-3*, pages 323–327, 2001.

[7] J. Torres-Jimenez, L. Vega-Garcia, C. A. Coutino-Gomez, and F. J. Cartujano-Escobar. Sstp: An approach to solve sat instances through partition. *Transactions on Computers*, 3(5):1482–1487, 2004.

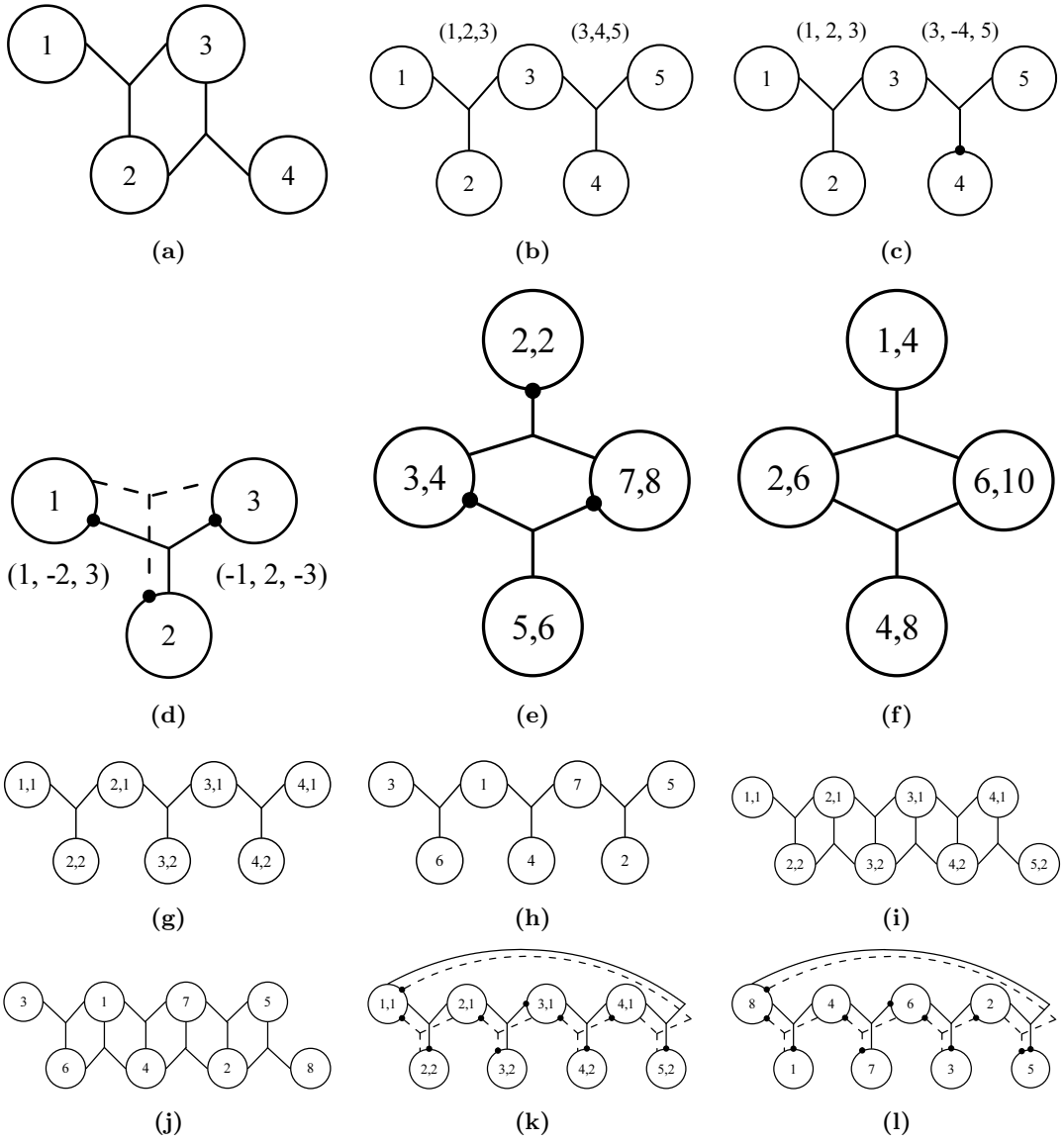

Figure 1: (a) A 3-uniform hypergraph. (b) A 3SAT instance represented with a 3-hypergraph. (c) A 3SAT instance represented with a σhypergraph. (d) A 3SAT instance represented with a σmultihypergraph. (e) $R = \{\{(2,2),(3,4),(5,6),(7,8)\},\{\{-(2,2),(3,4),(7,8)\},\{-(3,4),(5,6),-(7,8)\}\}$. (f) Graphical representation for the operation $S = (abs(R))_{-1,2}$. (g) Graphical representation of S from the $Algorithm1$. (h) Graphical representation of T from the $Algorithm2$. (i) Graphical representation of S from the $Algorithm3$. (j) Graphical representation of T from the $Algorithm3$. (k) Graphical representation of S from the $Algorithm4$. (l) Graphical representation of T from the $Algorithm4$.