Solving SAT By Computing A Stable Set Of Points In Clusters

Eugene Goldberg
email: eu.goldberg@gmai.com

***Abstract*—Earlier we introduced the notion of a stable set of points (SSP). We proved that a CNF formula is unsatisfiable iff there is a set of points (i.e. complete assignments) that is stable with respect to this formula. Experiments showed that SSPs for CNF formulas of practical interest are very large. So computing an SSP for a CNF formula point by point is, in general, infeasible. In this report[1], we show how an SSP can be computed in "clusters", each cluster being a large set of points that are processed "simultaneously". The appeal of computing SSPs is twofold. First, it allows one to better take into account formula structure and hence, arguably, design more efficient SAT algorithms. Second, SAT solving by SSPs facilitates parallel computing.**

I. Introduction

In [5], [7], we introduced the notion of a **stable set of points (SSP)** for CNF formulas. (By points here we mean complete assignments.) We showed that to prove F unsatisfiable it suffices to construct an SSP for F. If F is satisfiable, no SSP exists. The appeal of SSPs is twofold. First, they are formula specific, which allows one to exploit formula structure (e.g. formula symmetries). Second, an SSP can be viewed as a proof of unsatisfiability where different parts of this proof are related weakly. This facilitates parallel computing.

Even though for some classes of formulas there are polynomial size SSPs, in general, SSPs are exponential in formula size. A simple procedure for building an SSP "point by point" was given in [5]. Experiments showed that the number of points in an SSP grew very large even for small CNF formulas. This implies that building an SSP point by point is, in general, impractical. To address this problem, it was suggested in [5], [7] to compute an SSP "in clusters" thus processing many points simultaneously. In this report, we describe computing SSPs in clusters in greater detail.

The contribution of this report is fourfold. First, we introduce the notion of a **stable set of clusters** (**SSC**). The latter represents an SSP implicitly and can be computed much more efficiently. Although we introduce only the notion of clusters consisting of points, the stability of more complex objects (like clusters of clusters of points) can be studied. Second, we describe how the notion of an SSC works in testing the satisfiability of symmetric formulas (in particular, pigeon-hole formulas). Third, we show how one can build an SSC where clusters are specified by cubes. Fourth, we argue that computing an SSC facilitates parallel SAT solving.

This report is structured as follows. Section II recalls the notion of SSPs and gives relevant definitions. In Section III, we introduce the notion of a stable set of clusters. Section IV describes how a stable set of clusters is computed for symmetric formulas. In Section V we present Gen_SSC, a procedure for computing a stable set of clusters where clusters are cubes. A discussion of Gen_SSC is presented in Section VI. Sections VII and VIII provide some background and conclusions.

[1]The idea of computing SSPs in clusters was first presented in the technical report [6]. Since the latter is only available on the author's website we decided to publish a new version to make it more accessible. In this publication we introduced many changes. In particular, we changed the procedure Gen_SSC where clusters are represented by cubes to make it more practical. We also presented an example of how Gen_SSC works. Besides, we added a discussion of how computing SSPs in clusters benefits parallel SAT solving.

II. Recalling Stable Sets Of Points

In this section, we recall the notion of SSP introduced in [5] and give relevant definitions.

A. Definitions

Definition 1: Denote by $\boldsymbol{B}$ the set $\{0,1\}$ of values taken by a Boolean variable. Let X be a set of Boolean variables. An **assignment** to X is a mapping $X' \mapsto B$ where $X' \subseteq X$. If $X' = X$ this assignment is called **a complete one**. We will denote by $\boldsymbol{B^{|X|}}$ the set of complete assignments to X. A complete assignment to the variables of X is also called a **point** of $B^{|X|}$.

Definition 2: A **literal** of a Boolean variable x is either x itself or its negation. A disjunction of literals is called a **clause**. A formula that is a conjunction of clauses is said to be in the conjunctive normal form (**CNF**). A clause C is called **satisfied** by an assignment $\boldsymbol{p}$ if $C(\boldsymbol{p}) = 1$. Otherwise, the clause C is called **falsified** by $\boldsymbol{p}$. The same applies to a CNF formula and an assignment $\boldsymbol{p}$.

Definition 3: Let F be a CNF formula. The **satisfiability problem** (SAT for short) is to find an assignment satisfying all the clauses of F. This assignment is called a **satisfying assignment**.

Definition 4: Let $\boldsymbol{p} \in B^{|X|}$ be a point (*i.e.*, a complete assignment to X) falsifying a clause C. The **1-neighborhood of $\boldsymbol{p}$** with respect to C (written $\boldsymbol{Nbhd(p,C)}$) is the set of points satisfying C that are at Hamming distance 1 from $\boldsymbol{p}$.

It is not hard to see that the number of points in $Nbhd(\boldsymbol{p},C)$ equals that of literals in C.

Example 1: Let $C = x_1 \vee \overline{x}_3 \vee x_4$ be a clause specified in the Boolean space of 4 variables $x_1 \ldots, x_4$. Let $\boldsymbol{p} = (x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0)$ be a point falsifying

C. Then $Nbhd(\boldsymbol{p}, C)$ consists of the following three points: $\boldsymbol{p_1}$=($\boldsymbol{x_1 = 1}$, $x_2 = 1, x_3 = 1, x_4 = 0$), $\boldsymbol{p_2} = (x_1 = 0,\ x_2 = 1, \boldsymbol{x_3 = 0}, x_4 = 0)$, $\boldsymbol{p_3} = (x_1 = 0, x_2 = 1, x_3 = 1,\ \boldsymbol{x_4 = 1})$. Points $\boldsymbol{p_1}$, $\boldsymbol{p_2}$, $\boldsymbol{p_3}$ are obtained from $\boldsymbol{p}$ by flipping the value of variables x_1, x_3, x_4 respectively.

Definition 5: Given a formula F, denote by $\boldsymbol{Vars(F)}$ the set of its variables. Denote by $\boldsymbol{Z(F)}$ the set of complete assignments to $Vars(F)$ falsifying $\boldsymbol{F}$. If F is unsatisfiable, $Z(F) = B^{|X|}$ where $X = Vars(F)$.

Definition 6: Let F be a CNF formula and P be a subset of the set of falsifying points $Z(F)$. A function g mapping P to F is called a **transport function** if, for every $\boldsymbol{p} \in P$, the clause $g(\boldsymbol{p})$ is falsified by $\boldsymbol{p}$. In other words, a transport function $g : P \mapsto F$ is meant to assign each point $\boldsymbol{p} \in P$ a clause of F that is falsified by $\boldsymbol{p}$. We call the mapping $P \mapsto F$ above a transport function because it allows one to introduce some kind of "movement" of points in the Boolean space.

Definition 7: Let P be a nonempty subset of $Z(F)$ where F is a CNF formula. The set P is called **stable** with respect to F and a transport function $g : P \mapsto F$, if $\forall \boldsymbol{p} \in P$, $Nbhd(\boldsymbol{p}, g(\boldsymbol{p})) \subseteq P$. Henceforth, if we just say that a set of points P is stable with respect to a CNF formula F, we mean that there is a transport function $g : P \mapsto F$ such that P is stable with respect to F and g.

Example 2: Consider an unsatisfiable CNF formula F consisting of 7 clauses: $C_1 = x_1 \vee x_2$, $C_2 = \overline{x}_2 \vee x_3$, $C_3 = \overline{x}_3 \vee x_4$, $C_4 = \overline{x}_4 \vee x_1$, $C_5 = \overline{x}_1 \vee x_5$, $C_6 = \overline{x}_5 \vee x_6$, $C_7 = \overline{x}_6 \vee \overline{x}_1$. Clauses of F are composed of the six variables $x_1, \dots, x_6$. Let $P = \{p_1, \dots, p_{14}\}$ where $\boldsymbol{p_1} = 000000, \boldsymbol{p_2} = 010000, \boldsymbol{p_3} = 011000, \boldsymbol{p_4} = 011100, \boldsymbol{p_5} = 111100, \boldsymbol{p_6} = 111110, \boldsymbol{p_7} = 111111, \boldsymbol{p_8} = 011111, \boldsymbol{p_9} = 011011, \boldsymbol{p_{10}} = 010011, \boldsymbol{p_{11}} = 000011, \boldsymbol{p_{12}} = 100011, \boldsymbol{p_{13}} = 100010, \boldsymbol{p_{14}} = 100000$. (Values of variables are specified in the order variables are numbered. For example, $\boldsymbol{p_4}$= $(x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 0)$. The set P is stable with respect to the transport function g specified as: $g(\boldsymbol{p_1}) = C_1$, $g(\boldsymbol{p_2}) = C_2$, $g(\boldsymbol{p_3}) = C_3$, $g(\boldsymbol{p_4}) = C_4$, $g(\boldsymbol{p_5}) = C_5$, $g(\boldsymbol{p_6}) = C_6$, $g(\boldsymbol{p_7}) = C_7$, $g(\boldsymbol{p_8}) = C_4$, $g(\boldsymbol{p_9}) = C_3$, $g(\boldsymbol{p_{10}}) = C_2$, $g(\boldsymbol{p_{11}}) = C_1$, $g(\boldsymbol{p_{12}}) = C_7$, $g(\boldsymbol{p_{13}}) = C_6$, $g(\boldsymbol{p_{14}}) = C_5$. It is not hard to see that g indeed is a transport function i.e. for any point $\boldsymbol{p_i}$ of P it is true that $C(\boldsymbol{p_i}) = 0$ where $C = g(\boldsymbol{p_i})$. Besides, every point $\boldsymbol{p_i}$ of P satisfies the condition $Nbhd(\boldsymbol{p}, g(\boldsymbol{p})) \subseteq P$ of Definition 7. Consider, for example, point $p_{10} = 010011$. The value of $g(\boldsymbol{p_{10}})$ is C_2 where $C_2 = \overline{x}_2 \vee x_3$. The value of $Nbhd(\boldsymbol{p_{10}}, C_2)$ is $\{\boldsymbol{p_{11}} = 000011, \boldsymbol{p_9} = 011011\}$. So, the latter is a subset of P.

Proposition 1: . If there is a set of points that is stable with respect to a CNF formula F, then F is unsatisfiable.

The proof of this proposition is given in [7]. The reverse of Proposition 1 is true too *i.e.*, for every unsatisfiable formula F there is an SSP. A trivial SSP is $B^{|X|}$ where $X = |Vars(F)|$.

B. Procedure For Building SSP

In this subsection, we recall a simple procedure introduced in [5], [7] that generates an SSP point by point. We will refer to it as *Gen_SSP*. The pseudocode of *Gen_SSP* is shown in Figure 1. *Gen_SSP* accepts a CNF formula F and returns either a satisfying assignment or an SSP proving F unsatisfiable. *Gen_SSP* maintains two sets of points: *Boundary* and *Body*. The set *Boundary* (respectively *Body*) consists of the reached points whose neighborhood points have not been generated yet (respectively are already generated).

$Gen\_SSP(F)$ {
1 $\boldsymbol{p}_{init} := gen\_point(F)$
2 $Body = \emptyset, Boundary = \{\boldsymbol{p}_{init}\}$
3 while ($Boundary \neq \emptyset$) {
4 $\quad \boldsymbol{p} := pick\_next\_point(Boundary)$
5 $\quad Boundary := Boundary \setminus \{\boldsymbol{p}\}$
6 $\quad Body := Body \cup \{\boldsymbol{p}\}$
7 $\quad H = falsified\_clauses(F, \boldsymbol{p})$
8 $\quad$ if ($H = \emptyset$) return($\boldsymbol{p}, \emptyset$) // $\boldsymbol{p}$ is a satisf. assign.
9 $\quad C := pick\_clause(H)$ // $g(\boldsymbol{p}) := C$
10 $\quad NewPnts := Nbhd(\boldsymbol{p}, C) \setminus (Body \cup Boundary)$
11 $\quad Boundary := Boundary \cup NewPnts$}
12 return($nil, Body$) // $Body$ is an SSP, since $Boundary = \emptyset$

Fig. 1. Generation of SSP

The set *Boundary* is initialized with a point $\boldsymbol{p}_{init}$ while *Body* is originally empty (lines 1-2). Then, in a *while* loop (lines 3-11), *Gen_SSP* does the following. It picks a point $\boldsymbol{p}$ of *Boundary* to explore its neighborhood, removes $\boldsymbol{p}$ from *Boundary* and adds it to *Body* (lines 4-6). Then it computes the set H of clauses of F falsified by $\boldsymbol{p}$. If H is empty, $\boldsymbol{p}$ is a satisfying assignment. So, *Gen_SSP* returns $\boldsymbol{p}$ and an empty set indicating that no SSP is built (line 8). Otherwise, a clause $C \in H$ is picked as the value of the transport function g at $\boldsymbol{p}$ in line 9 (*Gen_SSP* builds g on the fly). The points of $Nbhd(\boldsymbol{p}, C)$ that are not in *Body* yet and not already in *Boundary* are added to *Boundary* (lines 10-11).

If the set *Boundary* is empty, it means that for every point $\boldsymbol{p} \in Body$, the property $Nbhd(\boldsymbol{p}, g(\boldsymbol{p})) \subseteq Body$ holds. So, *Body* is an SSP and hence F is unsatisfiable (line 12).

III. Computing A Stable Set Of Clusters

In this section, we introduce the notion of a stable set of clusters of points. As we mentioned earlier, experiments show that computing an SSP point by point is impractical. Building a stable set of clusters can be viewed as a way to speed up SSP computing by processing many points at once.

Definition 8: Let F be a CNF formula and P be a subset of $Z(F)$. Let g be a transport function $Z(F) \mapsto F$. Denote by $\boldsymbol{Nbhd(P, g)}$ the union of sets $Nbhd(\boldsymbol{p}, g(\boldsymbol{p}))$, $\boldsymbol{p} \in P$ for all the points of P. In other words, $Nbhd(P, g)$ is the union of the 1-neighborhoods for all points $\boldsymbol{p} \in P$ where $Nbhd(\boldsymbol{p}, g(\boldsymbol{p}))$ is computed with respect to the clause $g(\boldsymbol{p})$.

Definition 9: Let F be a CNF formula and $P_1, \dots, P_k$ be subsets of $Z(F)$. Let g_i be a transport function $P_i \mapsto F$, $i = 1, \dots, k$. Suppose that for every $P_i, i = 1, .., k$ the property $Nbhd(P_i, g_i) \subseteq P_1 \cup \dots \cup P_k$ holds. Then the set $\{P_1, \dots, P_k\}$ will be called a **stable set of clusters (SSC)** with respect to F and transport functions $g_1, \dots, g_k$. (Here we refer to a subset P_i as a **cluster** of points.)

Proposition 2: Let F be a CNF formula and $P_1, \dots, P_k$ be a stable set of clusters with respect to F and transport functions $g_1, .., g_k$. Then $P_1 \cup \dots \cup P_k$ is an SSP and so F is *unsatisfiable*.

A *proof* of this proposition is given in the appendix. The same applies to all *new* propositions introduced in this paper.

Remark 1: Note that if P is an SSP for F, any set of k subsets $P_i \subseteq P$ forms an SSC if $P_1 \cup \dots \cup P_k = P$. However we are interested only in clusters that make computing an SSC efficient. Intuitively, such efficiency can be achieved if every cluster P_i is formed from the points that are somehow related to each other. More specifically, every cluster is supposed to satisfy the following two properties. First, P_i has a short description regardless of its size *e.g.*, if P_i is exponential in $|Vars(F)|$. (One can think of P_i as a set of low Kolmogorov complexity.) Second, the 1-neighborhood of P_i with respect to the transport function g_i can be easily computed.

The notion of SSC is important for a few reasons. Suppose for the unsatisfiable formulas of some class there is an SSC with a polynomial number of clusters (in formula size). Then one can have an efficient procedure for testing the satisfiability of the formulas of this class. We substantiate this idea by the example of pigeon-hole formulas. Another reason for studying SSCs is that they expose a deep relation between models and formulas. So, one can get a better understanding of the existing SAT algorithms (e.g. those based on clause learning).

In this report, we consider only a two-level "hierarchy" of clusters, namely, clusters consisting of points. However, one can introduce more complex hierarchies (*e.g.*, clusters of clusters of points and so on).

IV. Testing Satisfiability Of Symmetric Formulas

In this section, we show the relation between permutational symmetries of a formula and its SSCs. In Subsection IV-A, we recall the previous results on SSPs for symmetric formulas. Subsection IV-B shows that the procedure for solving symmetric formulas introduced in [7] can be actually interpreted as building an SSC. Finally, in Subsection IV-C, we consider SSCs for pigeon-hole formulas.

A. Stable sets of points for symmetric formulas

Definition 10: Let X be a set of Boolean variables. A **permutation** $\boldsymbol{\pi}$ defined on the set X is a bijective mapping of X onto itself.

Definition 11: Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. Let $\boldsymbol{p} = (x_1, \dots, x_n)$ be a point of $B^{|X|}$. Let π be a permutation of X. **Denote by** $\boldsymbol{\pi(p)}$ the point $(\pi(x_1), \dots, \pi(x_n))$.

Definition 12: Let $F = \{C_1, \dots, C_k\}$ be a CNF formula. Let π be a permutation of $Vars(F)$. Denote by $\pi(C_i)$ the clause obtained from C_i by replacing each variable $x_m \in Vars(C_i)$ with the variable $\pi(x_m)$. **Denote by** $\boldsymbol{\pi(F)}$ the set of clauses $\{\pi(C_1), \dots, \pi(C_k)\}$

Definition 13: Let F be a CNF formula and π be a permutation of $Vars(F)$. Formula F is called **symmetric** with respect to π if $\pi(F)$ consists of the same clauses as F (*i.e.*, each clause $\pi(C_i)$ of $\pi(F)$ is identical to a clause C_m of F).

Definition 14: Let X be a set of Boolean variables and G be a group of permutations of X. **Denote by** $\boldsymbol{symm(p, p', G)}$ the following binary relation between points of $B^{|X|}$. A pair of points $(\boldsymbol{p}, \boldsymbol{p'})$ is in $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ iff there is $\pi \in G$ such that $p' = \pi(\boldsymbol{p})$.The relation $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ is an equivalence one and so it breaks $B^{|X|}$ into equivalence classes.

Definition 15: Points $\boldsymbol{p}$ and $\boldsymbol{p'}$ of $B^{|X|}$ are called **symmetric** with respect to a group G of permutations of X if they are in the same equivalence class of $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$.

Proposition 3: Let X be a set of Boolean variables and $\boldsymbol{p}$ be a point of $B^{|X|}$. Let C be a clause falsified by $\boldsymbol{p}$. Let π be a permutation of X. Then for each point $\boldsymbol{p'}$ of $Nbhd(\boldsymbol{p}, C)$ there is a point $\pi(\boldsymbol{p'})$ of $Nbhd(\pi(\boldsymbol{p}), \pi(C))$.)

The proof is given in [7].

Definition 16: Let F be a CNF formula that is symmetric with respect to a group G of permutations of $X = Vars(F)$. Let P be a set of points of $B^{|X|}$ falsifying F. The set P is called **stable modulo symmetry** G with respect to F and a transport function $g : P \mapsto F$ if for each point $\boldsymbol{p} \in P$, every point $\boldsymbol{p'}$ of $Nbhd(\boldsymbol{p}, g(\boldsymbol{p}))$ is either in P or there is a point $\boldsymbol{p''}$ of P that is symmetric to $\boldsymbol{p'}$.

Proposition 4: Let F be a CNF formula, P be a set of points of $B^{|X|}$, $X = Vars(F)$, that falsify F. Let $g : P \mapsto F$ be a transport function. If P is stable modulo symmetry G with respect to F and g, then F is **unsatisfiable**.

The proof is given in [7].

B. Stable sets of clusters for symmetric formulas

Proposition 4 is proved in [7] via extending the set of points P by adding the points symmetric to those of P. The transportation function g is also extended as follows. If $\boldsymbol{p} \in P$ and $\boldsymbol{p'} = \pi(\boldsymbol{p})$, then $g(\boldsymbol{p'})$ is equal to $\pi(g(\boldsymbol{p}))$ (In other words, for symmetric points, the extended transport function g assigns symmetric clauses.) It is shown in [7] that this extended set of points is actually an SSP for F with respect to the extended transport function g.

Importantly, one can give a different interpretation of the extension of P above. Let $P = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_s}\}$. Let $E(\boldsymbol{p_i})$ be the equivalence class of the symmetry relation $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ consisting of the points of $B^{|X|}$ that are symmetric to $\boldsymbol{p_i}$. Then the set of clusters $E(\boldsymbol{p_1}), \dots, E(\boldsymbol{p_s})$ **form an SSC** because $E(\boldsymbol{p_1}) \cup \dots \cup E(\boldsymbol{p_s})$ is exactly the extended set of points described above and so this set is stable. (Note that if points $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ of P are symmetric, then $E(\boldsymbol{p_i}) = E(\boldsymbol{p_j})$.)

Sets $E(\boldsymbol{p_i})$ meet the two requirements to clusters specified by Remark 1 of Section III. On one hand, each cluster is an equivalence class of the relation $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ and so the set of points of $E(\boldsymbol{p_i})$ can be easily described. On the other hand, the set $Nbhd(E(\boldsymbol{p_i}), g)$ (where g is the transport function extended from the original function $P \mapsto F$ as described before) is easy to compute. According to Proposition 3, $Nhbd(\pi(\boldsymbol{p}), \pi(C))$ and $Nbhd(\boldsymbol{p}, C)$ consist of points symmetric under π. Let $Nbhd(\boldsymbol{p_i}, C) = \{\boldsymbol{p_{i_1}}, \dots, \boldsymbol{p_{i_m}}\}$ (here

C is the clause $g(\boldsymbol{p_i})$). Then $Nbhd(E(\boldsymbol{p_i}), g) = E(\boldsymbol{p_{i_1}}) \cup \dots \cup E(\boldsymbol{p_{i_m}})$.

The procedure for building an SSC for a CNF formula F with symmetry G is essentially identical to the procedure of [7] for building a set P that is stable with respect to F modulo symmetry G. In turn, the procedure of [7] is different from the one shown in Figure 1 only in one line of code (line 11). Namely, when building a set of points stable modulo symmetry G this procedure does not add to *Boundary* a point $\boldsymbol{p'}$ of $Nbhd(\boldsymbol{p}, C)$ if *Total* contains a point that is symmetric to $\boldsymbol{p'}$. Eventually this procedure builds a set of points $P = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_m}\}$ that is stable with respect to F modulo symmetry G.

Importantly, one can interpret the procedure of [7] as building an SSC equal to $\{E(\boldsymbol{p_1}), \dots, E(\boldsymbol{p_m})\}$. This procedure just uses points $\boldsymbol{p_i}$ of P as representatives of clusters $E(\boldsymbol{p_i})$. Suppose, for instance, that a point $\boldsymbol{p'}$ of $Nbhd(\boldsymbol{p}, C)$ is not added to *Boundary* because it is symmetric to a point $\boldsymbol{p''}$ of *Total*. In terms of SSCs this just means that $E(\boldsymbol{p'}) = E(\boldsymbol{p''})$ and so the cluster $E(\boldsymbol{p'})$ has already been "visited".

C. Stable sets of clusters for pigeon-hole formulas

In this subsection, we illustrate the power of SSCs by the example of pigeon-hole formulas. These are unsatisfiable CNF formulas that describe the pigeon-hole principle. Namely, if $n > m$, then n pigeons cannot be placed in m holes so that no two pigeons occupy the same hole. In [8] A. Haken showed that pigeon-hole formulas have only exponential size proofs in the resolution proof system, which makes them hard for the SAT-solvers based on resolution. Since the pigeon-hole principle is symmetric with respect to a permutation of holes or pigeons, pigeon-hole formulas are highly symmetric.

Let $PH(n,m)$ denote a CNF formula encoding the pigeon-hole principle above. Let G denote the permutational symmetry of $PH(n,m)$. In [7] we showed that there is a set of points $P = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_{2m+1}}\}$ that is stable for $PH(n,m)$ modulo symmetry G. Denote by $S(n,m)$ the union of the equivalence classes $E(\boldsymbol{p_i})$, $i = 1, \dots, 2m+1$ of the relation $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$. The fact that P is stable modulo symmetry G means that $S(n,m)$ is an SSP for $PH(n,m)$. On the other hand, this fact means that $PH(n,m)$ has an SSC consisting of $2m+1$ clusters $E(\boldsymbol{p_i})$. The size of $E(\boldsymbol{p_i})$ is exponential in m and hence $S(n,m)$ is exponential in m too. However, the size of the SSC above in terms of clusters is **linear** in m.

V. Computing SSCs Using Cubes As Clusters

In this section, we introduce *Gen_SSC*, a SAT procedure that builds a special class of SSCs where clusters are cubes. Subsections V-A and V-B provide some definitions and an example of how *Gen_SSC* operates. In Subsection V-C, we present the pseudocode of *Gen_SSC*.

A. A few more definitions and examples

Definition 17: Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. A **cube** P of $B^{|X|}$ is a subset of $B^{|X|}$ that can be represented as $B_1 \times \dots \times B_n$, where B_i is a non-empty subset of B and $'\times'$ means the Cartesian product. The components B_i equal to $\{0\}$ or $\{1\}$ are called **literal components** of P.

Definition 18: We will say that a cube P **satisfies** (respectively **falsifies**) a clause C if every point $\boldsymbol{p} \in P$ satisfies (respectively falsifies) C.

Definition 19: Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. Let $P = B_1 \times \dots \times B_n$ be a cube of $B^{|X|}$ and B_i be equal to $\{0,1\}$. Let P', P'' be the cubes obtained from P by replacing the set B_i with sets $\{0\}$ and $\{1\}$ respectively. We will say that cubes P' and P'' are obtained from P by **splitting** on the variable x_i.

Definition 20: Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. Let C be a clause, $Vars(C) \subseteq X$. Denote by $\boldsymbol{Unsat(C)}$ the set of all points of $B^{|X|}$ that falsify C. It is not hard to see that $Unsat(C)$ **is a cube** of $B^{|X|}$.

Example 3: Let $C = x_2 \vee \overline{x}_4$ and $X = \{x_1, x_2, x_3, x_4\}$. Then $Unsat(C)$ equals $\{0,1\} \times \{0\} \times \{0,1\} \times \{1\}$. In other words, $Unsat(C)$ consists of all the points of $B^{|X|}$ for which $x_2 = 0$ and $x_4 = 1$. So, the second and fourth components of $Unsat(C)$ are literal components.

Definition 21: Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. Let $\boldsymbol{p}$ be a point of $B^{|X|}$. Denote by $\boldsymbol{Nbhd(p, x_i)}$ the neighborhood of $\boldsymbol{p}$ in direction x_i, i.e. the one-element set $\{\boldsymbol{p'}\}$ where the point $\boldsymbol{p'}$ is obtained from $\boldsymbol{p}$ by flipping the value of x_i in $\boldsymbol{p}$.

From Definition 4 and Definition 21, it follows that $Nbhd(\boldsymbol{p}, C)$ is the union of $Nbhd(\boldsymbol{p}, x_i)$ for all the variables of the clause C.

Definition 22: Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. Let $P = B_1 \times \dots \times B_n$ be a cube of $B^{|X|}$ and B_i be equal to $\{0\}$ or $\{1\}$. Denote by $\boldsymbol{Nbhd(P, x_i)}$ the union of $Nbhd(\boldsymbol{p}, x_i)$ for all the points $\boldsymbol{p}$ of P. It is not hard to see that $Nbhd(P, x_i)$ is the cube obtained from P by replacing B_i with the set $\{0,1\} \setminus B_i$. We will call $Nbhd(P, x_i)$ the **1-neighborhood cube** of P in direction x_i.

Definition 23: We will say that a **cube** P **falsifies** a clause C if $P \subseteq Unsat(C)$. (Obviously, in this case, every point of P falsifies C.)

Definition 24: Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. Let P be a cube of $B^{|X|}$ and C be a clause falsified by P. Denote by $\boldsymbol{Nbhd(P, C)}$ the set of 1-neighborhood cubes $Nbhd(P, x_i)$ in every direction $x_i \in Vars(C)$.

Note that cubes satisfy the two requirements to clusters specified in Remark 1 of Section III. On one hand, the set of points contained in a cube P can be succinctly described. On the other hand, if a clause C is falsified by P, the 1-neighborhood $Nbhd(P, C)$ is the union of a small number of cubes. So it can be easily computed.

Definition 25: Clauses C', C'' are called **resolvable** on a variable x if they have the opposite literals of only one variable and this variable is x. The clause C is said to be obtained by **resolution** of C', C'' on x if it consists of all the literals of C', C'' but those of x. The clause C is also called the **resolvent** of C', C'' on x.

Definition 26: Let $P = B_1 \times .. \times B_n$ be a cube of $B^{|X|}$ where $X = \{x_1, .., x_n\}$. We will **represent** P as a **conjunction** of literals where the i-th literal component of P corresponds to the literal $l(x_i)$ of this conjunction and vice versa. Namely,

- $B_i = \{0\} \Leftrightarrow l(x_i) = \overline{x_i}$.
- $B_i = \{1\} \Leftrightarrow l(x_i) = x_i$.

For every assignment $\boldsymbol{p} \in P$ this conjunction evaluates to 1 and vice versa.

Example 4: Let $P = \{0,1\} \times \{0\} \times \{0,1\} \times 1$ be a cube of $B^{|X|}$ where $X = \{x_1, x_2, x_3, x_4\}$. Then P can be specified by the conjunction $\overline{x}_2 \wedge x_4$. For the sake of simplicity we **will omit** the sign $'\wedge'$. Then the cube P is **specified** as $\overline{x}_2\, x_4$.

Definition 27: Let C', C'' be two clauses resolvable on a variable $x_i \in X$. Let P', P'' be two cubes of $B^{|X|}$ that falsify C' and C'' respectively. (This implies that the i-th component of P' and P'' is $\{b\}$ and $\{\overline{b}\}$ respectively where $b \in \{0,1\}$.) Let C be the resolvent of C' and C'' on x_i. A cube P is said to be obtained by **merging** P', P'' on x_i if

- $P' \subseteq P$ and $P'' \subseteq P$
- P falsifies C

Example 5: Let $X = \{x_1, \dots, x_8\}$ and $C' = x_1 \vee x_3 \vee x_7$ and $C'' = \overline{x}_1 \vee x_7$. Let $P' = \overline{x}_1\, \overline{x}_3\, x_5\, \overline{x}_7\, x_8$ and $P'' = x_1\, \overline{x}_3\, x_5\, \overline{x}_7$ be cubes of $B^{|X|}$ falsifying the clauses C' and C'' respectively. Let $P = \overline{x}_3\, x_5\, \overline{x}_7$. Note that $P' \subseteq P$ and $P'' \subseteq P$. Besides, P falsifies the resolvent $C = x_3 \vee x_7$ of C' and C''. So P can be viewed as obtained by merging P' and P'' on x_1. Note that, in general, a cube satisfying the two conditions of Definition 27 **is not unique**. For instance, the cube $P = \overline{x}_3\, \overline{x}_7$ satisfies Definition 27 as well.

Definition 28: Let P be a cube and A be a set of cubes $\{P_1, \dots, P_k\}$. We will say that P is **covered** by A if $P \subseteq Union(A)$ where $\boldsymbol{Union(A)} = P_1 \cup \dots \cup P_k$. That is the set of points specified by P is a subset of the set of points specified by A.

B. An example of how Gen_SSC operates

In this subsection, we give an example of how Gen_SSC operates. Consider the formula $F(X) = C_1 \wedge \dots \wedge C_5$ where $C_1 = x_2 \vee x_3, C_2 = x_1 \vee \overline{x}_2, C_3 = \overline{x}_1 \vee \overline{x}_2 \vee x_3$, $C_4 = \overline{x}_3 \vee x_4, C_5 = \overline{x}_3 \vee \overline{x}_4$, $X = \{x_1, x_2, x_3, x_4\}$.

A fragment of the execution trace of Gen_SSC applied to F is shown in Fig. 2. (Appendix II provides the *complete* trace where building an SSC for F is finished proving the latter unsatisfiable.) Like Gen_SSP, Gen_SSC maintains the sets *Body* and *Boundary*. The difference is that these sets consist of **cubes** rather than points. Gen_SSC starts with picking an initial cube of the set *Boundary* (line 1). Assume that Gen_SSC picks the cube P_1 equal to $\overline{x}_2\, \overline{x}_3$ as the initial cube. P_1 falsifies the clause $C_1 = x_2 \vee x_3$. So, Gen_SSC adds $g(P_1) := C_1$ to the definition of the transport function g. At this point, $Body = \emptyset$ and $Boundary = \{P_1\}$ (line 2).

Then Gen_SSC computes $Nbhd(P_1, C_1)$ *i.e.*, the neighborhood of P_1 with respect to C_1 (lines 3-5). It consists of the cubes $P_2 = x_2\, \overline{x}_3$ and $P_3 = \overline{x}_2\, x_3$ obtained from P_1 by negating the literals of x_2 and x_3 respectively. P_1 is moved from *Boundary* to *Body* and P_2, P_3 are added to *Boundary*.

initialize:
1 $P_1 = \overline{x}_2\, \overline{x}_3$, $g(P_1) = C_1 = x_2 \vee x_3$
2 $Body = \emptyset, Boundary = \{P_1\}$
compute $Nbhd(P_1, C_1)$:
3 $P_2 = Nbhd(P_1, x_2) = x_2\, \overline{x}_3$
4 $P_3 = Nbhd(P_1, x_3) = \overline{x}_2\, x_3$
5 $Body = \{P_1\}, Boundary = \{P_2, P_3\}$
splitting P_2 on x_1
6 $P'_2 = \overline{x}_1\, x_2\, \overline{x}_3$, $g(P'_2) = C_2 = x_1 \vee \overline{x}_2$
7 $P''_2 = x_1\, x_2\, \overline{x}_3$, $g(P''_2) = C_3 = \overline{x}_1 \vee \overline{x}_2 \vee x_3$
8 $Body = \{P_1\}, Boundary = \{P'_2, P''_2, P_3\}$
merging cubes P'_2, P''_2:
9 $Merge(P'_2, P''_2, x_1) = P_2 = x_2\, \overline{x}_3$
10 $C_6 = Res(C_2, C_3, x_1) = \overline{x}_2 \vee x_3$
11 $Body = \{P_1\}, Boundary = \{P_2, P_3\}$,
12 $F = F \wedge C_6$, $g(P_2) = C_6$
compute $Nbhd(P_2, C_6)$:
13 $Nbhd(P_2, x_2) = \overline{x}_2\, \overline{x}_3 = P_1$ and $P_1 \in Body$
14 $P_4 = Nbhd(P_2, x_3) = x_2\, x_3$
15 $Body = \{P_1, P_2\}, Boundary = \{P_3, P_4\}$,
....

Fig. 2. A fragment of an execution trace of Gen_SSC

Assume Gen_SSC picks P_2 to replace it in *Boundary* with the 1-neighborhood cubes. Since P_2 does not falsify any clause of F, Gen_SSC cannot immediately compute the 1-neighborhood of P_2. So, Gen_SSC splits it on x_1 replacing P_2 with cubes $P'_2 = \overline{x}_1\, x_2\, \overline{x}_3$ and $P''_2 = x_1\, x_2\, \overline{x}_3$ (lines 6-8). Note that P'_2 falsifies C_2 and P''_2 falsifies C_3.

To compute the 1-neighborhood of P_2, Gen_SSC merges P'_2 and P''_2 on x_1. This merging reproduces P_2 and generates a new clause falsified by it (lines 9-12). It is not hard to show that P_2 indeed satisfies Definition 27. First, $P'_2 \subseteq P_2$ and $P''_2 \subseteq P$. Second, P_2 falsifies the new clause C_6 obtained by resolving C_2 and C_3 (falsified by P'_2 and P''_2 respectively).

Now Gen_SSC is able to compute $Nbhd(P_2, C_6)$ (lines 13-15). The 1-neighborhood cube in direction x_2 is covered by *Body* since this cube equals to P_1 and $P_1 \in Body$. So, it is not added to *Boundary*. On the other hand, $Nbhd(P_2, x_3)$ is a new cube P_4 that is added to *Boundary*. As we mentioned above, the rest of the execution trace is given in Appendix II.

C. Procedure for building an SSC using cubes as clusters

In this subsection, we present the pseudocode of Gen_SSC (Figure 3). Gen_SSC accepts a formula F and returns a satisfying assignment or an SSC proving F unsatisfiable. As we mentioned above, like Gen_SSP, Gen_SSC maintains sets *Boundary* and *Body*. Here *Boundary* (respectively *Body*) is the set of cubes whose 1-neighborhood cubes have not been generated yet (respectively are already added to *Boundary*).

Gen_SSC starts with producing a cube P_{init} (line 1) to initialize the set $Boundary$ (line 2). $Body$ is initially empty. An SSC is built in a *while* loop (lines 3-22). First, a cube P is picked and removed from $Boundary$ (lines 4-5). Then the set H of clauses falsified by P is formed *i.e.*, for every clause C of H, it is true that $P \subseteq Unsat(C)$ (line 6).

After that, Gen_SSC checks if H is empty (line 7). If so, there are the two possibilities below. First, for every clause C of F it is true that $Unsat(C) \cap P = \emptyset$. This means that every

```
Gen_SSC(F) {
1  P_init := gen_cube(F)
2  Body = ∅, Boundary = {P_init}
3  while (Boundary ≠ ∅) {
4    P := pick_next_cube(Boundary)
5    Boundary := Boundary \ {P}
6    H = falsified_clauses(F, P)
7    if (H = ∅) {
8      if (satisfies(P, F)) // every p ∈ P satisfies F
9        return(P, ∅)
10     x := pick_var(P, F)
11     (P′, P″) := split_cube(P, x)
12     Boundary := Boundary ∪ uncov(P′, P″, Total)
13     continue }
14   (C′, P′, Merged) := merge_cubes(P, Boundary, F)
15   if (|Merged| > 1) { // merging is successful
16     Boundary := (Boundary \ Merged) ∪ {P′}
17     F := F ∧ C′
18     continue }
19   C := pick_clause(H) // g(P) := C
20   NewCubes := uncov(Nbhd(P, C), Total)
21   Boundary := Boundary ∪ NewCubes
22   Body := Body ∪ {P}}
23 return(nil, Body) // Body is an SSC => F is unsatisfiable
```

Fig. 3. Generation of SSC

point $\boldsymbol{p} \in P$ satisfies F (line 8). So, Gen_SSC returns P and an empty SSC (line 9). The second possibility is that there are clauses C of F such that $Unsat(C) \cap P \neq \emptyset$, but none of them is falsified by the cube P. In that case, Gen_SSC splits the cube P (lines 10-11) on a variable x into cubes P' and P''. Both cubes are tested by the function *uncov*, if they are covered by *Total* (see Definition 28) *i.e.*, whether P' or P'' is a subset of $Union(Total)$. Here $\boldsymbol{Total = Body \cup Boundary}$ and $\boldsymbol{Union(Total)}$ is the union of the cubes of *Total*. If P' or P'' is not covered by *Total*, it is added to $Boundary$ (line 12). Checking if P' or P'' is covered can be done by a regular SAT-solver (see the discussion of Subsection VI-B).

If H is not empty, Gen_SSC invokes the procedure $merge\_cubes$ (line 14). It tries to merge the cube P with other cubes of $Boundary$ to reduce the size of the latter. To this end, $merge\_cubes$ applies multiple merge operations described by Definition 27. It returns a) the subset $Merged$ of $Boundary$ consisting of the merged cubes including the cube P and b) a cube P' obtained by merging the cubes of $Merged$ and c) a new clause C' falsified by P' that is obtained by resolving clauses of F falsified by cubes of $Merged$. If $merge\_cubes$ succeeds, $|Merged| > 1$. Then the merged cubes are removed from $Boundary$, P' is added to $Boundary$ (line 16) and C' is added to F (line 17). Then a new iteration begins.

If $merge\_cubes$ fails, Gen_SSC picks a clause C of H (line 19) and forms the set of cubes $Nbhd(P, C)$. The function $uncov$ discards every cube of $Nbhd(P, C)$ covered by *Total* (line 20). The cubes of $Nbhd(P, C)$ that have not been discarded are added to $Boundary$ (line 21). Finally, P is added to $Body$ and a new iteration of the loop begins.

If *Boundary* is empty, then *Body* is an SSC. Gen_SSC returns the latter as a proof that F is unsatisfiable (line 23).

VI. Discussion Of Gen_SSC

In this section, we discuss Gen_SSC. In Subsection VI-A we give propositions stating that Gen_SSC is sound and complete. Subsection VI-B discusses potential improvements of Gen_SSC. In Subsection VI-C, we argue that Gen_SSC facilitates parallel solving.

A. Gen_SSC is sound and complete

Gen_SSC terminates when it builds a cube P satisfying every clause of F (line 9) or when the set *Boundary* is empty (line 23). The latter means that the set of points *Union*(*Body*) forms an SSP. In the first case F is correctly reported as satisfiable and in the second case it is properly identified as unsatisfiable. So, the proposition below holds. (As mentioned earlier, proofs of the new propositions are given in the appendix.)

Proposition 5: If Gen_SSC terminates, it returns the correct answer *i.e.*, Gen_SSC is *sound*.

One can also show that Gen_SSC is complete. Here is a high-level explanation why. The function $\xi = |Union(Body)| + |F|$ cannot decrease its value during the operation of Gen_SSC. That is ξ either grows or keeps its value unchanged. For instance, if Gen_SSC moves a cube from *Boundary* to *Body* the value of ξ increases. The same occurs after a new clause is generated and added to F when Gen_SSC merges cubes of *Boundary*. In the proof of completeness of Gen_SSC we show that the number of steps where ξ *preserves* its value is finite. This observation and the fact that the range of ξ is finite too implies that Gen_SSC always terminates. Hence, the proposition below holds.

Proposition 6: Gen_SSC terminates for every CNF formula *i.e.*, Gen_SSC is *complete*.

B. Improvements to Gen_SSC

The main flaw of the version of Gen_SSC described in Fig. 3 is as follows. Let P^* be either a cube obtained by splitting a cube of *Boundary* or a 1-neighborhood cube of a cube of *Boundary*. To find out if P^* needs to be added to *Boundary*, Gen_SSC checks if *Total* covers P^* (lines 12 and 20). That is, if P^* is a subset of $Union(Body \cup Boundary)$. This check can be performed by an "auxiliary" SAT solver based on conflict clause learning [10], [11]. The problem however is that such a check can be computationally hard.

There are at least two methods to address this problem. The first method is to make the auxiliary SAT solving easy by checking only if P^* is covered by a small subset of $Body \cup Boundary$. For instance, this subset may include only cubes sharing literal components with P^*. The other method is to combine regular SAT solving based on conflict clause learning and computing an SSC [3]. This method avoids auxiliary SAT solving at the expense of building an SSC specifying a larger SSP. Importantly, even if Gen_SSC builds an SSC specifying the trivial SSP equal to $B^{|X|}$, the SAT algorithm remains "local" since it does not produce a "global" certificate of unsatisfiability *i.e.*, an empty clause. So, in a sense, the size of the SSP specified by SSC does not matter.

C. Parallel SAT computing

Creating efficient algorithms of parallel SAT solving is a tall order [9]. One of the main problems here is that the SAT procedures used in practice prove unsatisfiability by resolution. A resolution proof can be viewed as **global** in the sense that a) it has a global goal (derivation of an empty clause) and b) different parts of the proof strongly depend on each other. This makes resolution procedures hard to parallelize.

On the other hand, an SSP can be viewed as a **local** proof. Given a formula F, one just needs to find a set of points P and a transport function $g : P \mapsto F$ such that a *local* property holds. Namely, for every point $\boldsymbol{p} \in P$, the relation $Nbhd(\boldsymbol{p}, C) \subseteq P$ holds where $C = g(\boldsymbol{p})$. Note that in contrast to a resolution proof, generation of an SSP does not have a global goal. So, arguably, building an SSP is easier to parallelize. Importantly, constructing an SSC produces a local proof as well since one simply builds an SSP *implicitly* (via clusters of points). So, one can argue that generating an SSC facilitates parallel computing too.

VII. Some Background

In this section, we briefly discuss the relation of SSPs and SAT algorithms based on local search (Subsection VII-A) and, in particular, the derandomized version of the Shöning procedure (Subsection VII-B). Besides, we relate SSCs with two "local" proof systems we introduced earlier (Subsection VII-C).

A. Local search procedures

SAT algorithms based on local search have been a subject of study for a long time. First, local search was applied only to satisfiable formulas. Papadimitriou showed [12] that a very simple stochastic local search procedure finds a satisfying assignment of a 2-CNF formula in polynomial time. Then, a few practical SAT-algorithms based on stochastic local search were developed and successfully applied to more general classes of satisfiable CNF formulas [14]. In [13] a new powerful stochastic algorithm for solving satisfiable CNF formulas was introduced by Shöning. Later, a derandomized version of that algorithm was developed that achieved the best known upper bound on complexity of solving k-SAT [2]. We will refer to this procedure as *Schn_der* where *Schn* stands for *Shöning* and *der* for *derandomized*. Importantly, *Schn_der* can be applied to both satisfiable and unsatisfiable CNF formulas.

On the one hand, SSPs can be related to local search algorithms. In particular, the Gen_SSP procedure recalled in Subsection II-B looks similar to *Schn_der* (see the the next subsection). On the other hand, the definition of an SSP is algorithm independent, which makes SSPs a very appealing object of study and separates them from the local search algorithms. This distinction becomes more conspicuous in this report where we consider the notion of a stable set of clusters. For example, the Gen_SSC procedure where clusters are cubes of points (see Section V) does not look like a local search procedure at all.

B. Schn_der and SSPs

In this subsection, we compare *Schn_der* and Gen_SSP computing an SSP point by point. Let F be a formula to check for satisfiability. *Schn_der* consists of two steps. First, *Schn_der* computes a set of Boolean balls covering the entire search space $B^{|X|}$ where $X = Vars(F)$. A **Boolean ball** with a center $\boldsymbol{p}$ and radius r is the set of all points $\boldsymbol{p'}$ such that $0 \leq distance(\boldsymbol{p}, \boldsymbol{p'}) \leq r$. (Here, *distance* specifies the *Boolean distance*.) Second, for every Boolean ball, *Schn_der* runs a procedure $Search(F, \boldsymbol{p}, r)$ that checks if this ball contains a satisfying assignment.

Gen_SSP can be viewed [1] as a version of *Schn_der* covering the space $B^{|X|}$ with balls of radius $r = 1$. Indeed, given a point $\boldsymbol{p}$ and a clause C falsified by $\boldsymbol{p}$, checking the 1-neighborhood $Nbhd(\boldsymbol{p}, C)$ mimics what the call $Search(F, \boldsymbol{p}, 1)$ does. The main difference between Gen_SSP and *Schn_der* is that, in contrast to the latter, the Boolean balls of the former *talk to each other*. This allows Gen_SSP to claim F to be unsatisfiable as soon as the set of visited balls becomes stable. Importantly, the idea of reaching the stability of talking Boolean balls can be extended to a huge variety of clusters of points (see Remark 1). Moreover, as we mentioned earlier, one can extend the notion of stability to multi-level clusters (e.g. clusters of clusters of points). The clustering of points serves here two purposes. First, it allows one to speed up SAT solving. Second, it facilitates exploiting the structure of the formula at hand by making clusters formula-specific.

C. Relation to proof systems NE and NER

In [4], we introduced two "local" proof systems, *NE* and *NER*. These proof systems are based on the fact that if a CNF formula F is satisfiable, there always exists a satisfying assignment $\boldsymbol{p}$ that satisfies only one literal of some clause C of F. (In terms of this report, $\boldsymbol{p}$ is located in $Nbhd(Unsat(C), C)$ *i.e.*, in the 1-neighborhood of C with respect to the cube $Unsat(C)$). The idea of either proof system is to explore the 1-neighborhood of all the clauses of F. The difference between *NE* and *NER* is that the latter allows one to use resolution to generate new clauses.

Let F be equal to $C_1 \wedge \cdots \wedge C_k$. One can show that Gen_SSC generates proofs similar to those of *NE* and *NER* if the set *Boundary* is initialized with the cubes $Unsat(C_i), i = 1, \ldots, k$. More precisely, *NE* is similar to Gen_SSC without the option of merging cubes of *Boundary* (and thus without the option of resolving clauses) and *NER* is similar to Gen_SSC if cube merging is allowed. The main flaw of *NE* and *NER* is that if F contains a small unsatisfiable subset of clauses, a proof in *NE* and *NER* still involves *all* clauses of F. (The notion of an SSP was actually designed to address this flaw of *NE* and *NER*.) On the other hand, in the case above, Gen_SSC can produce an SSC that involves only a small fraction of clauses of F.

VIII. Conclusions

Earlier we introduced the notion of a stable set of points (SSP) for a CNF formula F. (Here, a point is a complete assignment to the variables of F.) A CNF formula F is satisfiable if and only if it has a stable set of points. In this paper we present the notion of a stable set of clusters (SSC) of points. The main goal of using SSCs is to speed up the construction of an SSP for F by processing many points at once. We give two methods of computing SSCs. In the first method, clusters are specified by equivalence classes describing permutational symmetries of F. This method is an example of an algebraic approach to SAT solving. Importantly, one can extend the notion of a stable set of two-level clusters (*i.e.*, clusters of points) to multi-level ones (*e.g.*, clusters of clusters of points). In the second method, clusters are represented by cubes. In contrast to the first method that can be applied only to formulas with permutational symmetries, this method can be used for any CNF formulas. In addition to direct SAT solving, the second method can be employed for better understanding and improving the performance of existing SAT algorithms based on resolution.

REFERENCES

[1] E. Dantsin. A private communication.

[2] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k-sat based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.

[3] E. Goldberg. SAT solving by efficient computing of a stable set of clusters (a tentative title). A technical report to be published.

[4] E. Goldberg. Proving unsatisfiability of CNFs locally. *J. Autom. Reason.*, 28(4):417–434, 2002.

[5] E. Goldberg. Testing satisfiability of CNF formulas by computing a stable set of points. In *Proc. of CADE-02*, pages 161–180, 2002.

[6] E. Goldberg. Solving satisfiability problem by computing stable sets of points in clusters. Technical Report CDNL-TR-2005-1001, Cadence Berkeley Labs, 2005.

[7] E. Goldberg. Testing satisfiability of CNF formulas by computing a stable set of points. *Annals of Mathematics and Artificial Intelligenc*, 43(1-4):2005, January 65-89.

[8] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[9] Y. Hamadi and C. Wintersteiger. Seven challenges in parallel sat solving. AAAI'12, page 2120–2125. AAAI Press, 2012.

[10] J. Marques-Silva and K. Sakallah. Grasp – a new search algorithm for satisfiability. In *ICCAD-96*, pages 220–227, 1996.

[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *DAC-01*, pages 530–535, New York, NY, USA, 2001.

[12] C. H. Papadimitriou. On selecting a satisfying truth assignment. In *32nd Annual Symposium of Foundations of Computer Science*, pages 163–169, Oct 1991.

[13] T. Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science*, pages 410–414, 1999.

[14] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, page 337–343.

APPENDIX I
PROOFS OF PROPOSITIONS

Proposition 2: Let F be a CNF formula and $P_1,\dots,P_k$ be a stable set of clusters with respect to F and transport functions $g_1,..,g_k$. Then $P_1 \cup \dots \cup P_k$ is an SSP and so F is unsatisfiable.

Proof: Denote by P the set $P_1 \cup \dots \cup P_k$. Let g be a transport function such that for every $\boldsymbol{p} \in Z(F)$, it is true that $g(\boldsymbol{p}) = C$, where $C \in F$ and $C = g_i(\boldsymbol{p})$, $1 \le i \le k$. In other words, g assigns to $\boldsymbol{p}$ the same clause that is assigned to $\boldsymbol{p}$ by a function g_i (picked arbitrarily from $g_1,\dots,g_k$). Then P is an SSP with respect to F and the transport function g. Indeed, let $\boldsymbol{p}$ be a point of P and g_i be a transport function such that $g(\boldsymbol{p}) = g_i(\boldsymbol{p}) = C$. Since $\{P_1,\dots,P_k\}$ is an SSC, then $Nbhd(P_i, g_i) \subseteq P$. Hence $Nbhd(\boldsymbol{p}, g_i(\boldsymbol{p})) \subseteq P$ and so $Nbhd(\boldsymbol{p}, g(\boldsymbol{p})) \subseteq P$.

Proposition 5: If Gen_SSC terminates, it returns the correct answer *i.e.*, Gen_SSC is *sound*.

Proof: Let F be a CNF formula to test for satisfiability. Gen_SSC returns the answer *satisfiable* (line 9 of Fig. 3) only if an assignment satisfying F is found. So the answer *satisfiable* is always correct.

Now we show that if Gen_SSC reports that F is unsatisfiable (line 23), the set *Body* is an SSC for F. So the answer *unsatisfiable* is also always correct. Let P be a cube of *Body* and C be the clause assigned to P by the transport function g *i.e.*, $g(P) = C$ and P falsifies C. Originally, P appears in the set *Boundary* and is moved to *Body* only when the cubes of $Nbhd(P, C)$ are generated (lines 19-22).

Let P' be an arbitrary cube of $Nbhd(P, C)$. Let us show that P' will be covered by the final set *Body* and so the latter is an SSC for F. Indeed, if P' is not covered by the current set $Body \cup Boundary$, it is added to *Body* and hence it will be present in the final set *Body*. If P' is covered by the set $Body \cup Boundary$, there are two cases to consider. If P' is covered by *Body* alone, this means that every point of P' is already in *Union*(*Body*). So P' will be covered by the final set *Body*. If P' is *not* covered by *Body*, then some points of P' are present only in the current set *Boundary*. Since eventually *Boundary* becomes empty, the cubes containing those points of P' will be moved to *Body*. So, again, P' will be covered by the final set *Body*.

Proposition 6: Gen_SSC terminates for every CNF formula *i.e.*, Gen_SSC is *complete*.

Proof: Assume the contrary *i.e.*, Gen_SSC does not terminate on a CNF formula F. Consider the function $\xi = |Union(Body)| + |F|$. Note that ξ cannot reduce its value. That is, in every iteration of the loop of Gen_SSC, this value either stays the same or increases due to the growth of $|Union(Body)|$ and/or $|F|$. Since the maximum value of ξ is $2^n + 3^n$ (where $n = |Vars(F)|$), Gen_SSC can have only a finite set of iterations of the loop in which ξ grows. Since, by our assumption, Gen_SSC does not terminate, there exists an infinite sequence of iterations in which ξ preserves its value. Let us show that this is not the case.

The value of ξ does not change only when the cube P picked from *Boundary* is split or when every cube of $Nbhd(P, C)$ is covered by *Total*. (Recall that $Total = Body \cup Boundary$.) In the first case, P is replaced in *Boundary* with two cubes of a smaller size. In the second case, P is just removed from *Boundary*. The number of splits performed on a cube and its descendants is bound by 2^n. So, the total number of splits of the cubes of *Boundary* is limited by $3^n * 2^n$ where 3^n is the upper bound on $|Boundary|$ (because 3^n is the total number of different cubes of n variables). The total number of initialize:

1 $P_1 = \overline{x}_2\,\overline{x}_3,\ g(P_1) = C_1 = x_2 \vee x_3$

2 $Body = \emptyset, Boundary = \{P_1\}$

compute $Nbhd(P_1, C_1)$:

3 $P_2 = Nbhd(P_1, x_2) = x_2\,\overline{x}_3$

4 $P_3 = Nbhd(P_1, x_3) = \overline{x}_2\,x_3$

5 $Body = \{P_1\}, Boundary = \{P_2, P_3\}$

splitting P_2 on x_1

6 $P'_2 = \overline{x}_1\,x_2\,\overline{x}_3,\ g(P'_2) = C_2 = x_1 \vee \overline{x}_2$

7 $P''_2 = x_1\,x_2\,\overline{x}_3,\ g(P''_2) = C_3 = \overline{x}_1 \vee \overline{x}_2 \vee x_3$

8 $Body = \{P_1\}, Boundary = \{P'_2, P''_2, P_3\}$

merging cubes P'_2, P''_2:

9 $Merge(P'_2, P''_2, x_1) = P_2 = x_2\,\overline{x}_3$

10 $C_6 = Res(C_2, C_3, x_1) = \overline{x}_2 \vee x_3$

11 $Body = \{P_1\}, Boundary = \{P_2, P_3\}$

12 $F = F \wedge C_6,\ g(P_2) = C_6$

compute $Nbhd(P_2, C_6)$:

13 $Nbhd(P_2, x_2) = \overline{x}_2\,\overline{x}_3 = P_1$ and $P_1 \in Body$

14 $P_4 = Nbhd(P_2, x_3) = x_2\,x_3$

15 $Body = \{P_1, P_2\}, Boundary = \{P_3, P_4\}$,

....

splitting P_3 on x_4:

16 $P'_3 = \overline{x}_2\,x_3\,\overline{x}_4,\ g(P'_3) = C_4 = \overline{x}_3 \vee x_4$

17 $P''_3 = \overline{x}_2\,x_3\,x_4,\ g(P''_3) = C_5 = \overline{x}_3 \vee \overline{x}_4$

18 $Body = \{P_1, P_2\}, Boundary = \{P'_3, P''_3, P_4\}$

merging cubes P'_3, P''_3:

19 $P_3 = Merge(P'_3, P''_3, x_4) = P_3 = \overline{x}_2\,x_3$

20 $C_7 = Res(C_4, C_5, x_4) = \overline{x}_3$

21 $Body = \{P_1, P_2\}, Boundary = \{P_3, P_4\}$,

22 $F = F \wedge C_7,\ g(P_3) = C_7$

compute $Nbhd(P_3, C_7)$:

23 $Nbhd(P_3, C_7) = \overline{x}_2\,\overline{x}_3 = P_1$ and $P_1 \in Body$

24 $Body = \{P_1, P_2, P_3\}, Boundary = \{P_4\}$,

compute $Nbhd(P_4, C_7)$:

25 $Nbhd(P_4, C_7) = x_2\,\overline{x}_3 = P_2$ and $P_2 \in Body$

26 $Body = \{P_1, P_2, P_3, P_4\}, Boundary = \emptyset$

finish:

27 return *Body*

Fig. 4. Example of how Gen_SSC operates

iterations that remove a cube from *Boundary* without adding it to *Body* is also limited by 3^n. So, the total number of iterations that do not change the value of ξ is limited by $(2^n + 1) * 3^n$. Before this limit is exceeded, an event below takes place.

- A cube obtained by splitting satisfies all clauses of F and Gen_SSC terminates.
- A cube of $Nbhd(P, C)$ that is not covered by *Total* is added to *Body* thus increasing the value of ξ.
- A new clause is produced and added to F when merging cubes of *Boundary*, which increases the value of ξ.
- *Boundary* becomes empty and Gen_SSC terminates reporting that *Body* is an SSC and thus F is unsatisfiable.

In every case above, Gen_SSC either terminates or the value of ξ increases. So, Gen_SSC cannot have an infinite sequence of iterations where ξ does not change its value. Hence, Gen_SSC always terminates.

Appendix II
An example of how Gen_SSC operates

In this appendix, we complete the example of Subsection V-B. Namely, we describe the part of the execution trace after the dotted line (lines 16-27, Fig. 4). At this point, $Body = \{P_1, P_2\}$ and $Boundary = \{P_3, P_4\}$.

Gen_SSC picks $P_3 = \overline{x}_2\,x_3$ from *Boundary* and splits it on variable x_4 (lines 16-18). The reason for splitting is that P_3 does not falsify any clause of F. On the other hand, the cubes P'_3 and P''_3 produced by splitting falsify clauses C_4 and C_5 respectively. P_3 is replaced in *Boundary* with P'_3 and P''_3.

Then Gen_SSC merges cubes P'_3 and P''_3 to generate the cube P_3 again (lines 19-22). But now a new clause $C_7 = x_3$ is added to F that is falsified by P_3. The clause C_7 is produced by resolving clauses C_4 and C_5 falsified by P'_3 and P''_3. The cubes P'_3, P''_3 are replaced in *Boundary* with P_3.

Gen_SSC picks the cube P_3 again but now it is able to compute its 1-neighborhood with respect to the clause C_7 (lines 23-24). Since C_7 has only one literal, $Nbhd(P_3, C_7)$ consists of only one cube. Since this cube equals P_1 that is already in *Body*, P_3 is just moved from *Boundary* to *Body* without adding anything to the former.

Finally, Gen_SSC picks P_4, the last cube of *Boundary*. Since P_4 falsifies C_7, Gen_SSC computes the 1-neighborhood of the former with respect to the latter. This 1-neighborhood consists of the cube equal to P_2 that is already in *Body*. So, Gen_SSC just moves P_4 to *Body*.

At this point the set *Boundary* is empty. This means that the current set *Body* is an SSC and F is unsatisfiable. Gen_SSC returns *Body* as a proof of unsatisfiability (line 27).