Solving Satisfiability Problem by Computing Stable Sets of Points in Clusters

Cadence Berkeley Labs
1995 University Ave., Suite 460, Berkeley, California,94704
phone: (510)-647-2825, fax: (510)-486-0205

CDNL-TR-2005-1001
September 2005

Eugene Goldberg (Cadence Berkeley Labs), egold@cadence.com

Abstract. Earlier we introduced the notion of a stable set of points (SSP) and showed that a CNF formula is unsatisfiable iff there is a set of points (i.e. complete assignments) that is stable. Experiments showed that SSPs of CNF formulas of practical interest are very big. So computing an SSP of a CNF formula point by point is, in general, infeasible. In this report, we show how an SSP can be computed in "clusters", each cluster being a large set of points that are processed "simultaneously".

1. Introduction

In [2][3] we introduced the notion of a stable set of points (SSP) of CNF formulas. We showed that a CNF formula is unsatisfiable if and only if there is a set of points (i.e. complete assignments) that is stable. This means that to prove unsatisfiability of a CNF formula F it is sufficient to construct an SSP. (If F is satisfiable, then a satisfying assignment is to be found when building an SSP of F.)

The main appeal of SSPs is that they allow one to make full use of formula's structural properties. In particular, SSPs give a "natural" way of traversing the search space. Usually, a procedure for search space traversal is influenced by the requirement to guarantee search completeness. For example, in the DPLL procedure [1] , that is the basis of many algorithms used in practice, search is organized as a binary tree. In reality, the search tree is used only to impose a linear order on the points of the Boolean space to avoid visiting the same point twice. However, this order may be in conflict with "natural" relationships between points of the Boolean space that are imposed by the CNF formula to be checked for satisfiability (for example, if this formula has some symmetries).

Even though for some classes of formulas there are SSPs of polynomial size, in general, SSPs are exponential in formula size (which is something to be expected unless NP=coNP). A simple procedure for building an SSP of a formula F was given in [2][3]. Experiments showed that the number of points grew very large even for small CNF formulas. This implies that building an SSP "point by point" as it is done by the procedure of [2][3] is, in general, impractical. Two ways of overcoming this problem were suggested. One way is to compute SSP with excluded directions. As it was shown in [3] one can drastically reduce the size of SSPs if some directions are excluded from searching. By building such an SSP one proves that a clause (whose literals are specified by the excluded directions) is implied by the original formula F. This clause can be added to F to reduce the size of the SSPs with excluded directions to be computed next. Eventually, a small SSP with no excluded directions will be obtained (if the original formula is unsatisfiable). Summing up, the idea of this approach is to replace the construction of one huge "monolithic" SSP with building a sequence of small SSPs with excluded directions.

The other (and even more promising) way to solve the problem of computing SSPs of exponential size was mentioned in [2][3]. The idea is to compute an SSP "in clusters" processing many points simultaneously In this report, we consider the problem of computing SSPs in clusters in greater detail. The contribution of this report is threefold. *First*, we introduce the notion of a stable set of clusters. The importance of this notion is that it facilitates taking into account formula's structural properties. In particular, by giving a SAT-solver some information on the type of clusters to use, one may get exponential speed-up. Although we introduce only the notion of clusters consisting of points, the stability of more complex objects (like clusters of clusters of points and so on) can be studied.

Second, we show how one can build a stable set of clusters where clusters are cubes. Interestingly, "local" proof systems introduced in [4], can be viewed as a special way of building such a stable set of clusters. *Third*, we describe how the notion of a stable set of clusters works in testing the satisfiability of symmetric formulas (in particular, pigeon-hole formulas).

It should be noted that SAT-algorithms based on local search (that is SAT-algorithms operating on complete assignments) have been studied for more than a decade [7][8]. Recently a new powerful randomized local search algorithm was introduced in [9]. A derandomized version of this algorithm was

given in [5]. In Section 3 we briefly discuss the relation between SSPs and SAT-algorithms based on local search.

This report is structured as follows. In Section 2 we recall the notion of SSPs and give other relevant definitions. Section 3 briefly discusses the relation of SSPs and local search SAT-algorithms. In Section 4 we introduce the notion of a stable set of clusters. Section 5 describes a procedure for computing a stable set of clusters where clusters are cubes. In Section 6 we show how a stable set of clusters is computed for symmetric formulas. We make some conclusions in Section 7.

2. Main definitions

In this section, we recall the notion of SSP introduced in [2][3] and give other relevant definitions.

Let F be a CNF formula. Denote by ***vars(F)*** the set of variables of F. Denote by B the set {0,1} of values taken by a Boolean variable. Denote by $B^{|X|}$ the set of complete assignments to the set of Boolean variables $X= vars(F)$. A complete assignment to the variables of X is also called a ***point*** of $B^{|X|}$.

Definition 1. A disjunction of literals (also called a ***clause***) C is called ***satisfied*** by a complete assignment (point) $\boldsymbol{p}$ if $C(\boldsymbol{p}) = 1$. Otherwise, clause C is called ***falsified*** by $\boldsymbol{p}$. Denote by ***vars(C)*** the set of variables of C.

Definition 2. Let F be a CNF formula. The ***satisfiability problem*** (SAT for short) is to find a complete assignment (point) satisfying all the clauses of F. This assignment is called a ***satisfying assignment***.

Definition 3. Let $\boldsymbol{p} \in B^{|X|}$ be a point falsifying a clause C. ***The 1-neighborhood of point p*** with respect to clause C (written $Nbhd(\boldsymbol{p},C)$) is the set of points that are at Hamming distance 1 from $\boldsymbol{p}$ and that satisfy C.

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

Example 1. Let $C=x_1+\sim x_3+x_6$ be a clause specified in the Boolean space of 6 variables $x_1,\ldots, x_6$. Let $\boldsymbol{p}=(x_1=0, x_2=1, x_3=1, x_4=0, x5=1, x_6=0)$ be a point falsifying C. Then $Nbhd(\boldsymbol{p},c)$ consists of the following three points: $\boldsymbol{p_1}=(\underline{x_1=1}, x_2=1, x_3=1, x_4=0, x5=1, x_6=0)$, $\boldsymbol{p_2}=(x_1=0, x_2=1, \underline{x_3=0}, x_4=0, x5=1, x_6=0)$, $\boldsymbol{p_3}=(x_1=0, x_2=1, x_3=1, x_4=0, x5=1, \underline{x_6=1})$. 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_6 respectively i.e. the variables whose literals are in C.

Definition 4. Denote by ***Z(F)* the set of falsifying points** i.e. points at which F takes value 0. Obviously, if F is unsatisfiable, $Z(F) = B^{|X|}$ where $X= vars(F)$.

Definition 5. 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 any $\boldsymbol{p} \in P$, clause $g(\boldsymbol{p}) \in F$ is falsified by $\boldsymbol{p}$. In other words, a transport function $g:P\rightarrow F$ is meant to assign each point $\boldsymbol{p}\in P$ a clause of F that is falsified by $\boldsymbol{p}$. We call mapping $P \rightarrow F$ a transport function because it allows one to introduce some kind of "movement" of points in the Boolean space.

Definition 6. Let P be a nonempty subset of $Z(F)$ and F be a CNF formula. Set P is called stable with respect to a CNF formula F and transport function g: $P\rightarrow F$, if $\forall\ \boldsymbol{p} \in P$, $Nbhd(\boldsymbol{p}, g(\boldsymbol{p})) \subseteq P$.

Henceforth, if we say that a set of points P is stable with respect to a CNF formula F without mentioning a transport function, we mean that there is a function $g:P \rightarrow F$ such that P is stable with respect to F and g.

Example 2. Consider an unsatisfiable CNF formula F consisting of 7 clauses: $C_1=x_1+x_2$, $C_2=\sim x_2+x_3$, $C_3=\sim x_3+x_4$, $C_4=\sim x_4+x_1$, $C_5=\sim x_1+x_5$, $C_6=\sim x_5+x_6$, $C_7=\sim x_6+\sim x_1$. Clauses of F are composed of literals of 6 variables: $x_1,\ldots, x_6$. The following 14 points form an SSP P: $\boldsymbol{p_1}$=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, $\boldsymbol{p_4}$ consists of assignments $x_1=0$, $x_2=1$, $x_3=1$, $x_4=1$, $x_5=0$, $x_6=0$.) 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 6. Consider, for example, point $\boldsymbol{p_{10}}$=010011. The value of $g(\boldsymbol{p_{10}})$ is C_2, $C_2=\sim x_2+x_3$ and the value of $Nbhd(\boldsymbol{p_{10}},C_2)$ is {$\boldsymbol{p_{11}}$=000011, $\boldsymbol{p_9}$=011011}, the latter being 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.

Proof is given in [3].

3. SSPs and local search

In this section, we briefly discuss the relation of SSPs and SAT-algorithms based on local search (subsection 3.1) and, in particular, the procedure of [5] (section 3.2).

3.1 SSPs and local search procedures

SAT-algorithms based on local search have been a subject of study for more than a decade. First, local search was applied only to satisfiable formulas. Papadimitriou showed [7] 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 [8]. In [9] a new powerful stochastic algorithm for solving satisfiable CNF formulas was introduced by Shöning. Later, a derandomized version of that algorithm was developed [5] that achieved the best known upper bound on complexity of solving k-SAT. Importantly, the procedure of [5] can be applied to both satisfiable and unsatisfiable CNF formulas.

On the one hand, an SSP can be viewed as a representative of the world of local search algorithms. In particular, the procedure for building an SSP point by point ([2][3]) looks very similar to a procedure of [5] (see the discussion in the next subsection). On the other hand, the definition of 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 procedure for building such a set when clusters are cubes (see Section 5) does not look like a local search procedure at all.

3.2 SSPs and procedure of [5]

In this subsection we compare two procedures. The first procedure introduced by us in [2],[3] (its description is given in subsection 5.2) computes an SSP of a CNF formula F point by point. The second procedure [5] searches for a satisfying assignment of F in a Hamming ball with a center $\boldsymbol{p}$ and radius r. Needless to say that the objectives of these two procedures are different. Our procedure is meant for building an SSP and testing the satisfiability of F is just a "by-product". The objective of the procedure of [5] is to test the satisfiability of F. Nevertheless, these procedures look very similar.

If one sets radius r to n, where n is the number of variables in F, the procedure of [5] works "almost" like our procedure when the set *Boundary* is initialized with the point $\boldsymbol{p}$. The only difference is that the set of points visited by the procedure of [5] is, in general, smaller than the set of points built by our procedure. When a point $\boldsymbol{p}'$ is reached by our procedure, all (new) points of $Nbhd(\boldsymbol{p}',C)$ are added to *Boundary* where C is a clause of F falsified by $\boldsymbol{p}'$. On the other hand, the procedure of [5] "ignores" all the points of $Nbhd(\boldsymbol{p}',C)$ that are closer to the center $\boldsymbol{p}$ than the point $\boldsymbol{p}'$.

So an SSP is "redundant" in comparison to the set of points visited by the procedure of [5] when looking for a satisfying assignment in the Hamming ball of radius n. However this is a very "fruitful" redundancy because it allows one to make testing the stability of a set of points P local. Namely, for every point $\boldsymbol{p}$ of P, one just needs to test if $Nbhd(\boldsymbol{p}, g(\boldsymbol{p}))$ is in P and no knowledge of a center point is necessary. In its turn, the locality of stability testing is crucial in "speeding up" the computation of SSPs by building them in clusters of points.

4. Computing SSP by building a stable set of clusters

In this section, we introduce the notion of a stable set of clusters. As we mentioned in the introduction, experiments show that computing an SSP point by point is impractical. A natural way to speed up the computation is to process many points at once.

Definition 7. Let F be a CNF formula and D be a subset of $Z(F)$ where $X = vars(F)$. Let g be a transport function $Z(F)\rightarrow F$. Denote by $\boldsymbol{Nbhd(D, g)}$ the union of sets $Nbhd(\boldsymbol{p}, g(\boldsymbol{p}))$, $\boldsymbol{p} \in D$ for all the points of D. In other words, $Nbhd(D, \boldsymbol{g})$ is the union of 1-neighborhoods of the points of D where 1-neighborhood $Nbhd(p, g(\boldsymbol{p}))$ of a point $\boldsymbol{p}$ is computed with respect to a clause C of F assigned by the transport function g.

Definition 8. Let F be a CNF formula and $D_1,\ldots, D_k$ be subsets of $Z(F)$ where $X=vars(F)$. Let g_i, i=1,..,k be a transport function $Z(F)\rightarrow F$. Suppose that for every D_i, i=1,..,k it is true that $Nbhd(D_i, g_i) \subseteq D_1 \cup \ldots \cup D_k$. Then the set $\{D_1,..,D_k\}$ will be called a ***stable set of clusters*** **(SSC)** with respect to F and transport functions $g_1,.., g_k$. (Here we refer to a subset D_i as ***a cluster***).

Proposition 2. Let F be a CNF formula and $D_1,..,D_k$ be a stable set of clusters with respect to transport functions $g_1,.., g_k$. Then F is unsatisfiable.

Proof. Denote by P the set $D_1 \cup \ldots \cup D_k$. Let g be a transport function such that for every $\boldsymbol{p}\in Z(F)$, it is true that $g(\boldsymbol{p}) = C$, $C \in F$ where $C=g_i(\boldsymbol{p})$. In other words, the function g assigns to $\boldsymbol{p}$ the same clause that is assigned to $\boldsymbol{p}$ by some transport function g_i (that is picked arbitrarily from $g_1,.., g_k$). 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 $\{D_1,..,D_k\}$ is an SSC, then $Nbhd(D_i,g_i) \subseteq P$. Hence $Nbhd(\boldsymbol{p}, g_i) \subseteq P$ and so $Nbhd(\boldsymbol{p}, g) \subseteq P$.

Note that if P is an SSP of F, then any set of k sets $D_i \subseteq P$ such that $D_1 \cup \ldots \cup D_k = P$ forms an SSC. However, we are interested only in using "natural" clusters. Informally, set D_i is a natural cluster if the set of points of D_i and the set of points $Nbhd(D_i, g_i)$ can be "easily" specified.

The importance of the notion of an SSC is twofold. *First*, an obvious advantage of operating on clusters is that if for the formulas of some class, there is an SSC of a polynomial number of clusters (in formula's size), one can have an efficient procedure for testing the satisfiability of these formulas.

Second, a less obvious advandage is that by providing information about the structure of clusters one can "pump" a lot of information into a SAT-solver. It is highly unlikely that there is an efficient *universal* algorithm for solving SAT. From a practical point of view, this implies that to improve a SAT-solver's chance to efficiently solve formulas of a class, this SAT-solver needs some information about structural properties of formulas of this class. So a practical SAT-solver should have a "communication channel" with the user. Since a cluster may consists of an exponential number of points, by advising on the "shape" of clusters, the user may provide a SAT-solver with an enormous amount of information. As we will see later, this is exactly the case for symmetric formulas, where the shape of clusters is derived from formula's structural properties namely its symmetries.

In this report, we consider only a two-level "hierarchy", namely, clusters consisting of points. However, one can introduce more complex hierarchies (like clusters of clusters of points and so on). Arguably, using such hierarchies can help in capturing CNF formula's structure.

5. Computing SSCs using cubes as clusters

In this section, we describe a special class of SSCs where clusters are cubes. In subsection 5.1 we give a few more definitions we need in this section. Subsection 5.2 recalls the procedure of [2][3] for building an SSP. In subsection 5.3 we describe a procedure for building an SSC with cubes as clusters. We show that this procedure is complete and sound in subsection 5.4. Subsection 5.5 discusses how different initialization choices affect the procedure of subsection 5.3. In subsection 5.6 we describe the relation between SSCs with cubes as clusters and "local" proof systems of [4]. Finally, in subsection 5.7, potential improvements of the procedure of subsection 5.3 are discussed.

5.1 A few more definitions

Definition 9. Let $X =\{x_1,.., x_n\}$ be a set of Boolean variables. A ***cube*** D of $B^{|X|}$ is a subset of $B^{|X|}$ that can be represented as $A_1\times..\times A_n$, where A_i is a non-empty subset of B and '$\times$' means the Cartesian product. The components A_i of D that are equal to $\{0\}$ or $\{1\}$ are called ***literal components*** of the cube D.

Definition 10. Let $X =\{x_1,.., x_n\}$ be a set of Boolean variables. Let $D =A_1\times..\times A_n$ be a *cube* of $B^{|X|}$ and A_i be equal to $\{0,1\}$. Let

D_0,D_1 be the cubes obtained from D by replacing the set A_i with sets {0} and {1} respectively. We will say that cubes D_0 and D_1 are obtained from D by splitting in variable x_i.

Definition 11. Let $X=\{x_1,..,x_n\}$ be a set of Boolean variables. Let C be a clause, $vars(C) \subseteq X$. Denote by ***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 $C=x_2+\sim x_4$ 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^4 for which $x_2=0$ and $x_4=1$.

Definition 12. Let $X=\{x_1,.., x_n\}$ be a set of Boolean variables. Let $\boldsymbol{p}$ be a point of $B^{|X|}$. Denote by $Nbhd(\boldsymbol{p}, x_i)$ the ***neighborhood of p in direction x_i***, i.e. a one-element set $\{\boldsymbol{p}'\}$ where point $\boldsymbol{p}'$ is obtained from $\boldsymbol{p}$ by flipping the value of x_i in $\boldsymbol{p}$.

From Definition 3 and Definition 12, it follows that $Nbhd(p,C)$ is the union of $Nbhd(p, x_i)$ for all the variables of the clause C.

Definition 13. Let $X=\{x_1, .., x_n\}$ be a set of Boolean variables. Let $D=A_1\times..\times A_n$ be a cube of $B^{|X|}$ and A_i be equal to {0} or {1}. Denote by $Nbhd(D, x_i)$ the union of $Nbhd(\boldsymbol{p}, x_i)$ for all the points $\boldsymbol{p}$ of D. It is not hard to see that $Nbhd(D, x_i)$ is the cube obtained from D by replacing the set A_i with the set $\{0,1\} \setminus A_i$.

Definition 14. We will say that ***cube D falsifies clause C*** if $D \subseteq Unsat(C)$. (Obviously, in this case, every point of D falsifies C.)

Definition 15. Let $X=\{x_1, .., x_n\}$ be a set of Boolean variables. Let D be a cube of $B^{|X|}$ and C be a clause falsified by D. Denote by $Nbhd(D,C)$ the union of cubes $Nbhd(D, x_i)$ for all $x_i \in vars(C)$.

Note that cubes are "natural" clusters according to the informal definition of Section 3. On the one hand, the set of points a cube D contains can be easily specified. On the other hand, if a clause C is falsified by D, the neighborhood $Nbhd(D,C)$ is the union of a small number of cubes. So it can be easily specified as well.

5.2 Procedure for building an SSP

A generic procedure for building an SSP of a CNF formula was given in [2][3]. Its pseudocode is shown in Figure 1. The idea of this procedure is very simple. It maintains two sets of points: *Boundary* and *Body*. The set *Boundary* (respectively *Body*) consists of the reached points whose neighborhood has not been explored yet (respectively has been already explored). The set *Boundary* is initialized with a starting point $\boldsymbol{p}$ while the set *Body* is empty originally (lines 1-2).

Then in the while loop (lines 3-12) the *Generate_SSP* procedure does the following. It picks a point $\boldsymbol{p}'$ of *Boundary* to explore the neighborhood of $\boldsymbol{p}'$, removes $\boldsymbol{p}'$ from *Boundary* and adds it to *Body* (lines 4-6). Then it computes the set F' of clauses of F that are falsified by $\boldsymbol{p}'$. If F' is empty, then $\boldsymbol{p}'$ is a satisfying assignment and the procedure stops (lines 8-9). Otherwise, a clause C of F' is picked as the value of a transport function g at $\boldsymbol{p}'$ at line 10 (transport function $\boldsymbol{g}$ is built by *Generate_SSP* on the fly). The points of $Nbhd(\boldsymbol{p}',C)$, that are not in the set *Body* yet (and so their neighborhood has not been explored yet) and not in *Boundary* already, are added to the set *Boundary* (line 11). If the set *Boundary* is empty, it means that for any point $\boldsymbol{p}' \in Body$ each point of $Nbhd(\boldsymbol{p}', g(\boldsymbol{p}'))$ is in the set *Body* and so the latter is an SSP and hence F is unsatisfiable.

```
Generate_SSP(F)
/* Total = Boundary ∪ Body */
1  {p = Generate_starting_point(F);
2   Body =∅, Boundary = {p},
3    while(Boundary ≠ ∅)
4     {p′= pick_next_point(Boundary);
5      Boundary = Boundary \ {p′};
6      Body = Body ∪ {p′}.
7      F′= find_falsified_clauses(F, p′);
8       if (F′ = ∅)
9          return('satisfiable');
10      C = pick_a_clause(F′); /* C = g(p′) */
11     Boundary = Boundary∪(Nbhd(p′, C) \Total);
12    } /* while */
13   return(Body);/ *Body is an SSP now */
14  }
```

Figure 1. Pseudocode of procedure for building SSP

5.3 Procedure for building an SSC using cubes as clusters

In this subsection, we describe the *Generate_SSC* procedure for building an SSC that uses cubes as clusters. The pseudocode of this procedure is shown in Figure 2. This procedure also maintains two sets, *Boundary* and *Body*. The set *Boundary* consists of cubes whose 1-neighborhood has not been generated yet. The set *Body* consists either of cubes whose 1-neighborhood has been already generated or cubes whose 1-neighborhood has been "inherited" by other cubes during the splitting operation (see below).

The *Generate_SSC* procedure generates a cube (line 1) to initialize the set *Boundary* (line 2). *Body* is empty initially. An SSC is built in the while loop (lines 3-24). First, a cube D' is picked from *Boundary* (line 4). It is removed from *Boundary* and added to *Body* (lines 5-6). Then the set F' of clauses falsified by cube D' is formed (i.e. clause C of F is included in F' if $D' \subseteq Unsat(C)$) . If F' is not empty, then a clause C of F' is selected (line 9) and the set of cubes $Nbhd(D',C)$ is formed. The function *Uncov* (line 10) discards every cube C^* of $Nbhd(D',C)$ that is a subset of *Total* (the latter is the union of all the cubes from $Boundary \cup Body$). The cubes of $Nbhd(D',C)$ that have not been discarded are added to *Boundary* (line 10) and the current iteration of the loop ends.

If F' is empty, there are two possibilities. The first possibility is that for every clause C of F it is true that $Unsat(C) \cap D' = \emptyset$. This means that any point $\boldsymbol{p}$ of D' is a satisfying assignment. In this case, the *Generate_SSC* procedure returns '***satisfiable***' (lines 12-13). The second possibility is that there are clauses C of F such that $Unsat(C) \cap D' \neq \emptyset$, but none of them is falsified by the cube D'. In that case the *Generate_SSC* procedure has two different options. The first option is to split cube D' (lines 15-17) in a variable x_i into cubes D'_0, D'_1. Either cube is tested if it is a subset of $Total \setminus D$ and if not, it is added to *Boundary*.

```
/* Total = Union(Boundary ∪ Body) */
Generate_SSC(F)
1  { D = Generate_starting_cube(F);
2   Body =∅, Boundary = {D},
3    while(Boundary ≠ ∅)
4    {D´= pick_next_cube(Boundary);
5      Boundary = Boundary \ {D´};
6     Body = Body ∪ {D´}.
7     F´= find_falsified_clauses(F, D´);
8     if (F´≠ ∅)
9       {C = pick_a_clause(F´);
10       Boundary = Boundary ∪ Uncov(Nbhd(D´,C),Total);
11        continue;}
12    if (all_clauses_sat(D´))
13         return('satisfiable');
14     if (split_cube_option)
15          {(D'0, D'1) = split_cube(D´, xi);
16          Boundary =Boundary ∪ Uncov(D´0, D´1,Total \ D');
17           continue;}
18      if (generate_clause_option)
19          {(answer,C)=generate_falsified_clause(D',F);
20             if (answer == 'satsfiable')
21                return('satisfiable');
22          Boundary = Boundary∪Uncov(Nbhd(D´,C),Total);
23           continue;}
24     }/* end of while */
25  return('unsatisfiable',Body);
26  }
```

Figure 2. Pseudocode of procedure for building SSC

The other option is to generate a clause C implied by F and falsified by D' (lines 18-23) Such a clause can be generated in the following way. The idea is to add to the CNF formula F the set F^u of unit clauses specifying the points contained in D'. (For example, if all the points of D have x_i=0, then the unit clause $\sim x_i$ should be added to F.) The satisfiability of the modified formula (denote it by F*) can be tested by any "regular" SAT-solver. If F* is satisfiable, then F is satisfiable as well and the *Generate_SCC* procedure stops (lines 20-21). If F* is unsatisfiable, then the clause C consisting of the literals used in the clauses of F^u is implied by F and falsified by D'. (Note, however, that one can remove from C the literals corresponding to the unit clauses of D^u that have not contributed to proving the unsatisfiability of F*. By reducing the number of literals in the generated clause C, one reduces the number of cubes in $Nbhd(D',C)$.) After generating clause C, the cubes of $Nbhd(D',C)$ that are not discarded by the function *Uncov* are added to *Boundary* (line 22).

Of course, solving formula F* may be a hard problem. To reduce the work to be done by the chosen SAT-solver one can impose some limit on its use of internal resources (e.g. on the number of leaves of the search tree). If this limit is exceeded, no clause falsified by D' is generated. Then the only choice left (in case no clause of F is falsified by D') is to split the cube D'.

5.4 *Generate_SSC* is sound and complete

In this subsection, we show that *Generate_SSC* is a sound and complete procedure.

Proposition 3. The *Generate_SSC* procedure is sound i.e. if it terminates it returns the right answer.

Proof. Let F be a CNF formula to be tested for satisfiability. *Generate_SSC* returns the answer '***satisfiable***' (lines 13,21) only if an assignment satisfying F is found. So the answer '*satisfiable*' is always correct.

Now we show that if *Generate_SSC* says that F is unsatisfiable (line 25) then the set *Union*(*Body*) (which is the union of all the cubes of *Body*) is an SSP of F. So the answer '***unsatisfiable***' is also always correct. Let $\boldsymbol{p}$ be a point of *Union*(*Body*). Every point of *Union*(*Body*) first appears in the set *Union*(*Boundary*). Let D' be a cube of *Boundary* containing the point $\boldsymbol{p}$. Since *Generate_SSC* returns the answer '***unsatisfiable***' only if *Boundary* is empty, the point $\boldsymbol{p}$ eventually leaves *Boundary*.

Let us show that if $\boldsymbol{p}$ leaves the set *Union*(*Boundary*) and the set *Union*(*Body*) does not contain $\boldsymbol{p}$, each point of $Nbhd(\boldsymbol{p},C)$ is either added to *Total* or is already there. Here C is either a clause of F, or is implied by F. In either case $C(\boldsymbol{p})$=0.

If D' is not the only cube of Boundary containing $\boldsymbol{p}$, then removal of D' from *Boundary* does not remove $\boldsymbol{p}$ from *Union*(*Boundary*). Let us assume that D' is the only cube of *Boundary* containing $\boldsymbol{p}$ and it is picked by *pick_next_cube* (line 4). Suppose that D' is split into D'_0, D'_1 and the cube D'_0 contains $\boldsymbol{p}$. By assumption, $\boldsymbol{p}$ is not in *Union*(*Body*) and D' is the only cube of *Boundary* containing $\boldsymbol{p}$. Then D'_0 is not a subset of *Total* \ D' and is added to *Boundary* (line 22). Hence $\boldsymbol{p}$ does not leave *Union*(*Boundary*).

So $\boldsymbol{p}$ first time leaves *Union*(*Boundary*) and appears in *Union*(*Body*) only if a clause C falsified by D' is found in F or generated by a SAT-solver (line 19). In that case, every cube of $Nbhd(D',C)$ that is not a subset of *Total* is added to *Boundary*. It means, that every point of $Nbhd(\boldsymbol{p},C)$ is either added to *Total* or is already there.

The set *Total* built by *Generate_SSC* can not reduce in size. This means that if a point $\boldsymbol{p}$ appeared in *Union*(*Body*) for the first time and $Nbhd(\boldsymbol{p},C) \subseteq Total$, the latter relation holds true until *Generate_SSC* terminates. Since eventually *Total* = *Union*(*Body*), for every point $\boldsymbol{p}$ of *Union*(*Body*) it is true that $Nbhd(\boldsymbol{p},C) \subseteq$ *Union*(*Body*) (where C is a clause of F or is implied by F and $C(\boldsymbol{p})$=0)). Hence *Union*(*Body*) is an SSP of a formula F* implied by F (F* is obtained by adding to F all the generated clauses). Then F* and so F are unsatisfiable.

Proposition 4. The *Generate_SSC* procedure is complete. That is it terminates for any CNF formula F.

Proof. Assume the contrary. Let F be a CNF formula such that *Generate_SSC* does not terminate. The set *Total* can not reduce in size. This means that in every iteration of the loop of *Generate_SSC*, the set *Total* either grows or stays the same. Since the maximum size of Total is 2^n (where n=|*vars*(F)|) *Generate_SSC* can have only a finite set of iterations of the main loop in which *Total* grows. This means that *Generate_SSC* should have an infinite sequence S of iterations of the main loop in which the set *Total* stays the same. Let us show that such an infinite sequence is impossible.

The set *Total* does not change only when the cube D' picked from *Boundary* is split or when every cube of $Nbhd(D',C)$ is a subset of *Total*. In the first case, D' is replaced in *Boundary* with

two cubes of a smaller size. In the second case, D' is just removed from *Boundary*. Now we build a function $\boldsymbol{H}$ with a finite range that monotonically decreases in each iteration that does not change *Total*. (The existence of $\boldsymbol{H}$ means that the infinite sequence *S* above is not possible.) The only argument of $\boldsymbol{H}$ is the set *Boundary*. The output of $\boldsymbol{H}$ is a vector $\boldsymbol{V}$ with *n* components $V_1,\ldots,V_n$ where V_i is the number of cubes of *Boundary* that have exactly *i* literal components. Let vectors $\boldsymbol{V}$ and $\boldsymbol{V'}$ be compared lexicographically, that is $\boldsymbol{V'} < \boldsymbol{V}$ iff $V_i < V_i'$ and *i* is the smallest component number where $\boldsymbol{V}$ and $\boldsymbol{V'}$ are different. If *Boundary'* is obtained from *Boundary* by removing a cube, then $\boldsymbol{H}$(*Boundary'*) < $\boldsymbol{H}$(*Boundary*). The same is true if *Boundary'* is obtained from *Boundary* by replacing a cube D' with two smaller cubes obtained by the splitting of D'.

5.5 Initialization of the set Boundary

The performance of the *Generate_SSC* procedure strongly depends on how one initializes the set *Boundary* (line 1 of Figure 2.). Suppose, for example, that *Boundary* is initialized with a cube *D* that contains only one point (i.e. *D* is a cube of the smallest size). Let *C* be a clause of *F* falsified by the point of *D*. Each cube of *Nbhd*(*D*,*C*) contains only one point as well. So the *Generate_SSC* procedure reduces to the procedure of Section 5.2 that builds an SSP point by point.

Now, suppose that *Boundary* is initialized with the cube equal to $B^{|X|}$ where *X*=*vars*(*F*) i.e. with the largest possible cube of $B^{|X|}$. In this case, the set *Total* is initialized to the entire Boolean space. Let D' be a cube of *Boundary* and *C* be a clause of *F* falsified by D'. Since the set *Total* cannot decrease in size, it stays equal to $B^{|X|}$. Then every cube of *Nbhd*(D',*C*) is a subset of *Total* and so is not added to the set *Boundary*. Suppose that *Generate_SSC* does not use the option of new clause generation. Then, if for the chosen cube D' of *Boundary* there is no clause *C* of *F* falsified by D', the only choice is to split D' into smaller cubes D'_0 and D'_1. If, say cube D'_0, falsifies a clause of *F* it is removed from *Boundary*. Otherwise D'_0 is returned to the set *Boundary*. So, *Generate_SSC* keeps splitting cubes of *Boundary* until each cube resulting from splitting is falsified by a clause of *F*. In other words, if *Boundary* is initialized with the cube $B^{|X|}$ (and no new clauses are generated), the cubes of *Boundary* can be viewed as "nodes" of a binary tree. The only difference from a regular binary search procedure is that branches are examined in an arbitrary order.

If *Boundary* is initialized with the cube equal to $B^{|X|}$, *Generate_SSC* does not build a non-trivial SSP. It just checks if the entire space $B^{|X|}$ is an SSP. So the most interesting case to study is when *Boundary* is initialized with a cube that is neither equal to $B^{|X|}$ nor contains only one point of $B^{|X|}$. In that case, one can hope to build a non-trivial SSP (i.e. different from $B^{|X|}$) at the same time speeding up computation by processing many points at once.

5.6 Relation to proof systems of [4]

In this subsection, we show that the procedure *Generate_SSC*, in a sense, generalizes the proofs systems of [4].

In the pseudocode of *Generate_SSC* shown in Figure 2, the set *Boundary* is initiated with a single cube and the set *Body* is initially empty. However, *Boundary* and *Body* can be initiated with any set of cubes of $B^{|X|}$, *X* = *vars*(*F*) satisfying the following two conditions: a) if $D \in Body$, then $D \subseteq Z(F)$ (recall that *Z*(*F*) is the set of points falsifying *F*); b) if $D \in Body$, then for every point $\boldsymbol{p} \in D$, there is a clause C_i of *F* such that *Nbhd*($\boldsymbol{p}$, C_i) $\in$ *Union*(*Boundary*). In other words, one can add to *Body* any cube consisting only of points falsifying *F*, if 1-neghborhood of the points of *D* with respect to some transport function is in the set *Boundary*.

Let $C_1,..,C_k$ be the clauses of *F*. Then one can initialize *Body* with cubes *Unsat*(C_1),..,*Unsat*(C_k) if *Boundary* is initalized with cubes *Nbhd*(*Unsat*(C_1),C_1),.., *Nbhd*(*Unsat*(C_k),C_k). If *F* is unsatisfiable, then *Unsat*(C_1) $\cup \ldots \cup$ *Unsat*(C_k) is equal to $B^{|X|}$. In other words, in this case, *Generate_SSC* builds a trivial SSP equal to $B^{|X|}$. However, constructing even such a trivial SSP may be beneficial for the following reason. Current state-of-the-art solvers are based on the resolution proof system. In this system, one needs to generate an empty clause as a "global certificate" of unsatisfiability of a CNF formula. In a sense, this certificate is the result of merging "local branches", which makes these branches interdependent. The *Generate_SSC* procedure does not have to merge "branches" to produce a global certificate. As soon as the set *Boundary* is empty we know that *F* is unsatisfiable. So *Generate_SSC* is "inherently local".

Note that if *Boundary* and *Body* are initialized as described above, there is no need to add new cubes to *Boundary*. Indeed, if D' falsifies *C* and *C* is implied by *F*, then *Nbhd*(D',*C*) is a subset of the union of neighborhoods *Nbhd*(*Unsat*(C_i),C_i),*i*=1,..,*k*. (Because this union contains all neighborhood points of *Z*(*F*)). So only two things can happen to the set *Boundary*. Either a cube D' is moved from *Boundary* to *Body* (if there is a clause *C* implied by *F* and falsified by D') or D' is split and moved from *Boundary* to *Body* and the two cubes produced in the split are added to *Boundary*.

In [4] we introduced two "local" proofs 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 that is in the 1-neighborhood of a clause *C* of *F*. (In the notation of this report, 1-neighborhood of *C* is the union of cubes of *Nbhd*(*Unsat*(*C*),*C*)). The idea of either proof system is to explore the 1-neighborhood of all the clauses of *F*. In is not hard to show that *Generate_SSC* can "simulate" proofs generated in NE and NER if the set *Body* is initialized with cubes *Unsat*(C_i),*i*=1,..,*k* and *Boundary* is initialized with cubes *Nbhd*(*Unsat*(C_i),C_i),*i*=1,..,*k*

The system NE does not use resolution to generate new clauses. It is equivalent to *Generate_SSC* without the option to generate new clauses. (In this case, if there is no clause of *F* falsified by D', the only thing *Generate_SSC* can do is to split D'.) In contrast to NE, the system NER allows one to use the resolution operation to generate new clauses. The proofs of NER are simulated by *Generate_SSC* if new clauses are allowed to be generated by a resolution based SAT-solver.

5.7 Improvements of *Generate_SSC*

The pseudocode shown in Figure 2 captures only main features of the procedure *Generate_SSC*. Below we list some potential improvements.

1) Since *Generate_SSC* uses the operation of cube splitting, the number of cubes in *Boundary* (and so *Body*) may grow

exponentially. An interesting way of mitigating this problem is to merge cubes of *Boundary*. The idea is to replace a set of k cubes $D_{i1},..,D_{ik}$ with the smallest cube D' containing each cube of the set. In a sense, each cube of the set *Boundary* corresponds to a "search branch". So merging cubes of *Boundary* is, in a way, merging branches. The objective of this merging is to reduce the number of branches to examine. Adding the merging operation does not effect the soundness of *Generate_SSC* but may compromise its completeness. The latter is due to the fact that combining cube merging and splitting may lead to looping. This looping can be prevented in many ways. One way is to use merging only if the cube D' (where D' is the result of merging) falsifies a clause C of F or such a clause is generated. In this case, *Generate_SSC* adds to *Boundary* cubes of $Nbhd(D',C)$ adds to *Body* cube D' and no splitting of D' occurs.

2) In *Generate_SSC* every cube D of the set $Nbhd(D',C)$ is checked if $D \subseteq Total$. (Similarly either cube obtained by splitting D' is checked if it is a subset of $Total \setminus D'$). This check reduces to solving an instance of SAT and so can be performed by a SAT-solver. To reduce the run time of this SAT-solver, one can impose a limit on the amount of computation e.g. number of branchings. If the SAT-solver reaches this limit when checking if $D \subseteq Total$, D is added to *Boundary*.

3) Any clause C generated by a SAT-solver (line 19 of Figure 2) can be added to the formula F. This way one gets more choices when picking a clause of F falsified by a cube D'.

6. Testing satisfiability of symmetric formulas

In this section we show the relation between formula's symmetry and SSCs. (In this report we consider only permutational symmetry.) In subsection 6.1 we recall the results of [3] on SSPs of symmetric formulas. In subsection 6.2 we show that the procedure for solving symmetric formulas introduced in [3] can be actually interpreted as building an SSC. Finally, in subsection 6.3, we apply the results of subsection 6.2 to solving pigeon-hole formulas.

6.1 Testing satisfiability of symmetric formulas by computing an SSP

Definition 16. Let X be a set of Boolean variables. A ***permutation*** π defined on set X is a bijective mapping of X onto itself.

Definition 17. Let $X=\{x_1,\ldots, x_n\}$ be a set of Boolean variables. Let $\boldsymbol{p}=(x_1,.., x_n)$ be a point of $B^{|X|}$. Let π be a permutation of X. ***Denote by*** $\boldsymbol{\pi(p)}$ the point $(\pi(x_1),\ldots, \pi(x_n))$.

Definition 18. Let $F=\{C_1,..,C_k\}$ be a CNF formula. Let π be a permutation of $vars(F)$. ***Denote by*** $\boldsymbol{\pi(C_i)}$ the clause obtained from C_i by replacing each variable $x_m \in C_i$ with the variable $\pi(x_m)$. ***Denote by*** $\boldsymbol{\pi(F)}$ the set of clauses $\{\pi(C_1),..,\pi(C_k)\}$

Definition 19. 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 (that is each clause $\pi(C_i)$ of $\pi(F)$ is identical to a clause C_m of F).

Definition 20. Let X be a set of Boolean variables and G be a group of permutations of X. ***Denote by symm*** $\boldsymbol{(p, p', G)}$ the following binary relation between points of $B^{|X|}$. A pair of points $(\boldsymbol{p}, \boldsymbol{p'})$ is in $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ if and only if there is $\pi \in G$ such that $\boldsymbol{p'} = \pi(\boldsymbol{p})$.The relation $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ is an equivalence relation and so it breaks $B^{|X|}$ into equivalence classes.

Definition 21. 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 5. 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 a point $\boldsymbol{q} \in Nbhd(\boldsymbol{p},C)$ be obtained from $\boldsymbol{p}$ by flipping the value of $x_i \in vars(C)$. Let π be a permutation of X and $\boldsymbol{p'} = \pi(\boldsymbol{p})$, $C' = \pi(C)$. Let $\boldsymbol{q'} \in Nbhd(\boldsymbol{p'},C')$ be obtained from $\boldsymbol{p'}$ by flipping the value of $\pi(x_i)$. Then $\boldsymbol{q'}=\pi(\boldsymbol{q})$. (In other words, if $\boldsymbol{p'} = \pi(\boldsymbol{p})$, $C' = \pi(C)$, then for each point q of $Nbhd(\boldsymbol{p},C)$ there is a point $\boldsymbol{q'}=\pi(\boldsymbol{q})$ of $Nbhd(\boldsymbol{p'},C')$.)

Proof is given in [3].

Definition 22. 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*** $\boldsymbol{G}$ with respect to F and a transport function g: $P \rightarrow F$ 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 6. Let F be a CNF formula, P be a set of points of $B^{|X|}$, $X=vars(F)$, that falsify F. Let g: $P \rightarrow F$ be a transport function. If P is stable modulo symmetry G with respect to F and g, then F is unsatisfiable.

Proof is given in [3].

6.2 Testing satisfiability of symmetric formulas by computing an SSC

Proposition 6 is proven in [3] by "extending" the set of points P by adding each point of $B^{|X|}$ that is symmetric to a point 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 [3] that this extended set of points is actually an SSP of F with respect to the extended transport function g.

Interestingly, one can give a different interpretation of the extension of P above. Let $P=\{\boldsymbol{p}_1,.., \boldsymbol{p}_s\}$. Let $D(\boldsymbol{p}_i)$ be the equivalence class of $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ consisting of the points of $B^{|X|}$ that are symmetric to $\boldsymbol{p}_i$. Then the set of clusters $D(\boldsymbol{p}_1),..,D(\boldsymbol{p}_s)$ form an SSC because $D(\boldsymbol{p}_1) \cup \ldots \cup D(\boldsymbol{p}_s)$ is exactly the extended set described above and so this set is stable. (Note that if points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ of P are symmetric, then $D(\boldsymbol{p}_i)=D(\boldsymbol{p}_j)$.)

Sets $D(\boldsymbol{p}_i)$ are "natural" clusters according to the informal definition given in Section 3. On the one hand, each cluster is an equivalence class of the symmetry relation $symm(\boldsymbol{p}, \boldsymbol{p'}, G)$ and so the set of points of $D(\boldsymbol{p}_i)$ can be easily specified. On the other hand, set $Nbhd(D(\boldsymbol{p}_i), g)$ (where g is the transport function extended from the original function $P \rightarrow F$ as described before) is easy to define. According to Proposition 5, if $\boldsymbol{p'} = \pi(\boldsymbol{p})$ and $C' = \pi(C)$, then sets $Nhbd(\boldsymbol{p'},C')$ and $Nbhd(\boldsymbol{p},C)$ consist of points symmetric under π. Let $Nbhd(\boldsymbol{p}_i,C) = \{\boldsymbol{p}_{i1},.., \boldsymbol{p}_{im}\}$ (here C is the clause $g(\boldsymbol{p}_i)$). Then $Nbhd(D(\boldsymbol{p}_i), g) = D(\boldsymbol{p}_{i1}) \cup .. \cup D(\boldsymbol{p}_{im})$.

The procedure for building an SSC for a CNF formula F with symmetry G is essentially identical to the procedure of [3] for building a set P that is stable with respect to F modulo

symmetry G. In its turn, this procedure of [3] 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 $\boldsymbol{p}''$. (In the procedure of Figure 1 we do not add $\boldsymbol{p}''$ only if *Total* already contains the point $\boldsymbol{p}''$ *itself*.) Eventually this procedure builds a set of points $P=\{\boldsymbol{p}_1,..,\boldsymbol{p}_m\}$ that is stable with respect to F modulo symmetry G. On the other hand, one can interpret the procedure of [3] as building an SSC $D(\boldsymbol{p}_1),..,D(\boldsymbol{p}_m)$. This procedure just uses points $\boldsymbol{p}_i$ of P as representatives of clusters $D(\boldsymbol{p}_i)$. In particular, when 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 $D(\boldsymbol{p})=D(\boldsymbol{p}'')$ and so the cluster $D(\boldsymbol{p}'')$ has been already "visited".

6.3 Testing satisfiability of pigeon hole formulas by computing an SSC

In this section, we illustrate the power of SSCs by the example of pigeon-hole formulas. These are unsatisfiable CNF formulas that, by means of propositional logic, describe the pigeon-hole principle. This principle is that if $n > m$, then n objects (pigeons) cannot be placed in m holes so that no two objects occupy the same hole. In [6] A. Haken showed that pigeon-hole formulas have only exponential size proofs in the resolution proof system, which makes these formulas hard for the SAT-solvers based on resolution.

Since the pigeon-hole principle is symmetric with respect to a permutation of holes or a permutation of pigeons, pigeon-hole formulas are highly symmetric. In [3] we showed that pigeon-hole formula $PH(n,m)$ has a stable set of points $S(n,m)$ that is the union of $2*m+1$ equivalence classes $D_1,\ldots,D_{2m+1}$ of the relation $symm(\boldsymbol{p},\boldsymbol{p}',G)$ where G is the permutational symmetry of $PH(n,m)$. The set $S(n,m)$ consists of an exponential size of points, so (some) equivalence classes D_i have exponential size. This means that pigeon-hole formula $HP(n,m)$ has an SSC that consists of $2*m+1$ clusters $D_1,\ldots,D_{2m+1}$. That is the size of this SSC is linear in the number of holes.

7. Conclusions

We introduced the notion of a stable set of clusters (SSC) . The main purpose of using SSCs is to speed up building a stable set of points. We gave two methods of computing SSCs. The first methods uses Boolean cubes as clusters. In the second method clusters are equivalence classes of a symmetry relation describing formula's symmetry.

References

[1] M.Davis, G.Logemann, D.Loveland. *A Machine program for theorem proving*. Communications of the ACM. -1962. -V.5. -P.394-397.

[2] E.Goldberg. *Testing satisfiability CNF formulas by computing a stable set of points.* Technical Report CDNL-TR-2001-1126, November 2001.

[3] E.Goldberg. *Testing Satisfiability of CNF Formulas by Computing a Stable Set of Points*. SAT-2002, May 6-9, Cincinnati, Ohio, USA. Published in Annals of Mathematics and Artificial Intelligence, 43 (1-4): 65-89, January 2005. Available at http://eigold.tripod.com/papers/ annals-2005.zip.

[4] E.Goldberg. *Proving Unsatisfiability of CNFs locally*. Journal of Automatic Reasoning, vol 28:417-434,2002. Available at http://eigold.tripod.com/papers/jar-nbhd.zip .

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

[6] A.Haken. *The intractability of resolution*. Theor. Comput. Sci. 39 (1985), 297-308.

[7] C.Papadimitriou. *On selecting a satisfying truth assignment*. Proceedings of FOC-91.

[8] B.Selman, H.Kautz, B.Cohen. *Noise strategies for improving local search.* Proceedings of AAAI-94.

[9] U.Shöning. *A probabilistic algorithm for k-SAT and constraint satisfaction problems.* In Proceedings of FOCS'99,pp. 410-414, 1999.