On multiple learning schemata in conflict driven solvers

Andrea Formisano[1] and Flavio Vella[2]

[1] Università di Perugia formis@dmi.unipg.it
[2] IAC-CNR and Sapienza Università di Roma vella@di.uniroma1.it

Abstract. In this preliminary paper we describe a general approach for multiple learning in conflict-driven SAT solvers. The proposed formulation of the conflict analysis task turns out to be expressive enough to reckon with different orthogonal generalizations of the standard learning schemata, such as the conjunct analysis of multiple conflicts, the generation of possibly interdependent learned clauses, the imposition of global optimality criteria.
We formalize the general learning problem as a search for a collection of vertex cuts in a directed acyclic graph. Optimality of the solution may be evaluated with respect to a given global objective function intended to encode search strategies and heuristics affecting the behavior of the solver. We also outline some algorithmical solutions by exploiting standard algorithms proposed to solve cut and multicut problems on DAGs.

1 Introduction

Most of the successful SAT solvers available nowadays originate from refinements of the DPLL procedure [11] and integrate powerful techniques such as *conflict driven learning* and non-chronological backtracking. As a matter of fact, the combination of suitable learning schemata with smart branching heuristics and efficient (Boolean) constraint propagation algorithms [4], remarkably improved the efficiency and effectiveness of modern SAT solvers. Analogous techniques, often migrated from SAT technology, have been exploited in developing solvers in various fields of Automated Reasoning, such as Answer Set Programming, Constraint Programming, and Satisfaction Modulo Theory (cf., among many, [13, 26, 23] and the references therein).

In what follows we focus on clause-based SAT solving, albeit similar arguments can be advanced concerning other kind of solvers. More specifically, we will consider the problem of determining the satisfiability of a set of clauses, built up from a collection of propositional variables. (As usual, a literal is a variable or its complement. Clauses are sets of literals.)

Let us briefly recall the main traits of a DPLL-like SAT solver. For a formal and detailed treatment the reader is referred to [4], among many). Given an instance of SAT (i.e., a set of clauses), a DPLL-like SAT solver proceeds by alternating decision steps and propagation phases. By making a decision, the solver assigns one propositional variable a truth value. Then, it propagates the effects of such a decision to (possibly) derive implied assignments. Each decision has a *decision level* associated to it and propagation takes place, within the current decision level, whenever all but one literals in a clause have been assigned false (with a slight abuse, let us call *unit* this kind of clause).

In order to find a satisfying assignment for the unit clause (and then, for the whole instance), the unassigned literal must be set true. The propagation stage continues as long as units are produced. Then, the decision level is increased and another decision is taken. The process stops as soon as a solution is found (namely, all variables have been consistently assigned) or a *conflict* is detected. Normally, a conflict arises when, through the propagation phase, all the literals in a clause become assigned to false. At this point a *conflict analysis* procedure derives a new *conflict clause* to be added to the instance. Then, some of the previously taken decisions are undone and the solver *backjumps* to a previous decision level, before continuing the search for a solution. The presence of the new clause drives subsequent propagation phases and prevents the solver from generating again the very same conflicting assignment.

In this paper we propose a schema for conflict analysis general enough to enable the conjunct analysis of several conflicts and the consequent generation of more that one learned clause. Multiplicity may arise not only from generating more than one conflict in a single propagation phase, or by admitting multiple decisions at each decision level, but also from concurrently running different instances of the above outlined procedure (each one performing a different visit of the solution space). Designing a global learning procedure in such a general context has several potential advantages. On the one hand, it might take advantage from the results of all searches in order to derive more effective conflict clauses. On the other hand, learned clauses convey knowledge exchange between the concurrent threads of the parallel solver.

The paper is organized as follows. After recalling the basic notions about conflict-driven SAT solving (Section 2), in Section 3 we formalize our general learning schema. Sections 4 and 5 concretize our proposal by introducing some algorithmical solutions. Finally, Section 6 provides some concluding remarks and hints for future development.

2 Conflict analysis and implication graphs

Let us consider in more detail the conflict analysis procedure described earlier. Following [34, 22], the dependencies between decided and propagated variables can be described by means of an *implication graph*. It is a directed acyclic graph (DAG, for short) in which vertices represent truth value assignments for literals. We will often identify a vertex with the literal it represents: a variable x assigned true (resp., false) is rendered by a vertex x (resp., $-x$). Moreover, given a literal x, let $\overline{x}$ denote its complement.

Edges express the reasons that lead the assignments. In particular, decided variables correspond to vertices having no incident edges. If a literal x has been assigned true by propagation, because of a unit clause $\{x, y_1, \ldots, y_k\}$, then the vertex x has each $\overline{y_i}$ as direct antecedents in the implication graph.

To simplify the following treatment, let us introduce a special kind of vertex. A *conflict vertex*, not corresponding to any variable, is introduced in the graph whenever a pair of contradictory assignments is produced for the same propositional variable x. A conflict vertex has exactly two antecedent vertices, representing two inconsistent value assignments for a variable.

In this setting, given an implication graph $G(V, E)$, we can identify the set of *source* vertices $S \subseteq V$ (corresponding to decisions) and the set of conflict vertices $T \subseteq V$.

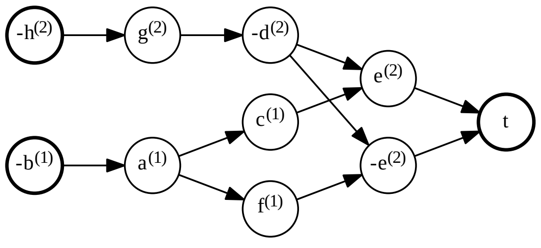


Fig. 1. Implication graph G_1 for Example 1.

Example 1. Consider the following clauses: $\{a, b\}$, $\{c, \neg a\}$, $\{\neg a, f\}$, $\{h, g\}$, $\{\neg g, \neg d\}$, $\{\neg f, d, \neg e\}$, $\{e, d, \neg c\}$. Fig. 1 shows an implication graph G_1 obtained by assigning true the two literals $\neg b$ and $\neg h$. Hence, two sources are present in G_1, corresponding to such decisions. (For the time being, let us ignore the decision levels indicated by the vertices' superscripts.) As mentioned, edges denote propagation steps. For instance, an edge from $\neg b$ to a is introduced because of the first clause. In fact, being b false, in order to satisfy the clause, a must be set true. Similarly, two edges from a, to vertices c and f, respectively, are introduced because of the second and the third clause, resp., and so on. Note that, once completed, the propagation steps introduces a pair of conflicting assignments for the literal e. A conflict vertex t denotes this fact. □

For simplicity, let us consider a graph G having a single conflict vertex t (having x and $\overline{x}$ as antecedents, for a variable x).[1] Every vertex cut X in G that separates S from t and such that $\{x, \overline{x}\} \not\subseteq X$, corresponds to a partial assignment sufficient to imply the conflict.[2] In other words, X can be translated into a conflict clause made of the complements of the literals in X.

Example 2. Consider the graph G_1 in Fig. 1. The vertex cut made of the vertices a and g separates all the sources from the conflict. Consequently, a partial assignment sufficient to imply the conflict consists in assigning true to both a and g.

Clearly, more that one vertex cut may exist. For instance, in G_1, two other possible vertex cuts are $\{a, \neg d\}$ and $\{c, f, \neg d\}$. □

If a vertex cut contains only one literal that has been assigned at a certain decision level ℓ, then such a literal is a *unique implication point* (UIP). Notice that, if s_ℓ is the literal decided at level ℓ, a UIP *dominates* (in the sub-graph of level ℓ) the conflict vertex t with respect to s_ℓ: each path from the source s_ℓ to t must go through the UIP.

The maximum decision level of the vertices in the cut (except the decision level of the conflict) is the decision level to backjump. After backjumping, the conflict clause becomes an *asserting clause*: all its literals but one are assigned false, so the remaining

[1] Such a situation can be always achieved by isolating a single conflict vertex t, and by restricting the implication graph to the *S−t-connected* portion of G.

[2] Let a and b be two distinct vertices such that b is reachable from a in a directed graph G. Recall that, a *vertex cut* (resp., *edge cut*) that separates a from b, is defined as a set of vertices (resp., edges) such that their removal from G eliminates all directed paths from a to b.

literal is determined by propagation. Consequently, the solver will be "guided" toward a different portion of the search space.

With respect to the same conflict, several UIPs may occur in the implication graph. These are ordered depending on the distance from the conflict vertex: the UIP closest to the conflict vertex is called *first UIP*, and so on.

Clearly, not all the possible vertex cuts involve a UIP. Nevertheless, each of them identifies a different conflict clause that can be, in principle, profitably added to the SAT instance in order to prune the search space.

Example 3. Consider again the graph G_1 in Fig. 1 and assume that $\neg b$ has been assigned first, at the decision level 1. (Decision levels are denoted by superscripts in the graph.) All the vertices assigned by propagation as a consequence of this decision belong to such decision level. Then, a subsequent decision, at level 2, assigned true to the literal $\neg h$ and caused the propagation of the other vertices. Hence, the conflict occurs at level 2. The first UIP is $\neg d$. Indeed it dominates t (each path from $\neg h$ to the conflict must go through the vertex $\neg d$) and it is the dominator closest to t. The procedure proceeds by learning the clause corresponding to such UIP, namely $\{\neg f, \neg c, d\}$, and backjumping to level 1 (hence, the decision regarding $\neg h$ is undone). In this situation, a propagation step, using the newly introduced clause forces d to be assigned true. In turn, because of clauses $\{\neg g, \neg d\}$ and $\{h, g\}$ also $\neg g$ and h are set true. □

3 Generalizing conflict analysis and learning

The effectiveness of a solver largely depends on the strategy used to identify suitable conflict clauses, among all the possible candidates, by analyzing the implication graph. Notice that, the structure of the implication graph substantially affects the learning procedure, because, as mentioned, conflict clauses correspond to vertex cuts. In turn, the structure of the graph depends on:

(a) the selection of decision variables (branching strategy). This determines the set of sources of the graph.

(b) The specific sequence of propagations performed in each propagation phase. If different asserting clauses are activable at each time, different orders of their activation induce different topologies of the graph.

Many of the existing solvers perform just a single decision at each step and stop the propagation phase as soon as the first conflict arises. Moreover, conflict analysis is usually focused on UIPs (often, only conflict clauses involving the first-UIP are learned). One reason for this choice is that such conflict clauses can be obtained by developing a linear derivation by propositional resolution. The steps of this derivation use as resolvents (in the reverse order) the asserting clauses used along the path from the (first) UIP to the conflict. Hence, in some sense, once the conflict is present, the generation of the conflict clause is deterministic [4, 34].

In what follows we generalize the basic learning schema described so far. More specifically, we design a general schema for conflict analysis that takes into account the following aspects:

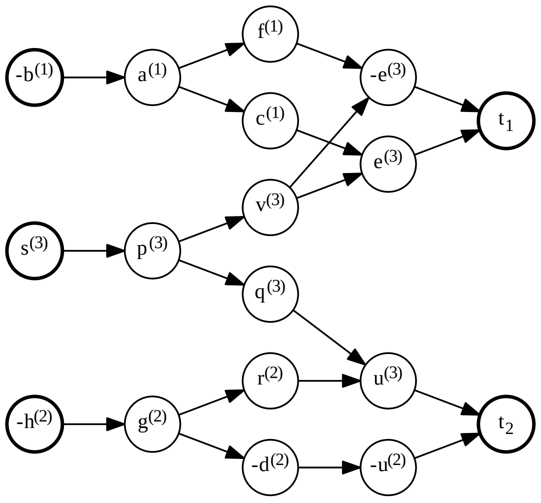


Fig. 2. Implication graph G_2 for Example 4.

- Multiple decisions taken at each decision level. This corresponds to focusing the search for a solution on a more restricted portion of the search space. This increases the effect of subsequent propagation phases. In addition, this splitting of the search space may be exploited by parallel solvers.
- Multiple conflicts. These might be generated both because of multiple decisions or because the propagation phase does not necessarily end when the first conflict is produced. For example, it might proceed until no further inference is possible.
- Weights associated to vertices and/or edges of the implication graph. This enables the application of some kind of heuristics in selecting conflict clauses. We will not enter into the details of how these weights are assigned. For the time being, it suffices observing that weights may be generated in various manners, depending on the involved heuristics. For instance, one could consider static measures, concerning the structure of the SAT instance at hand, or dynamic parameters such as the "relevance" exhibited by each variable during the previous part of the computation (as an immediate example, think about exploiting criteria somewhat akin to those used by branching strategies).

Most of the learning schemata appeared in the literature essentially consider only cuts involving UIPs. In this context, some comparative empirical evaluation of different learning schemata can be found in [34] and in [27]. Very few proposals concern multiple clauses learning. We just mention here the interesting work [17], that explores the advantage of learning several clauses from the same conflict.

Example 4. Consider this set of clauses: $\{a, b\}$, $\{c, \neg a\}$, $\{\neg a, f\}$, $\{h, g\}$, $\{\neg g, \neg d\}$, $\{e, \neg v, \neg c\}$, $\{\neg f, \neg v, \neg e\}$, $\{q, \neg p\}$, $\{v, \neg p\}$, $\{r, \neg g\}$, $\{d, \neg u\}$, $\{p, \neg s\}$, $\{u, \neg r, \neg q\}$. Suppose we proceed by deciding a single literal in each decision level and we develop an implication graph by first deciding $\neg b$ to be set true (decision level 1). After propagating the effect of this decision (literals a, c, f are set true, cf., the implication graph G_2 in Fig. 2), we step to decision level 2 and set true the literal $\neg h$. Again, a propagation phase is executed and this determines the truth values of the literals $g, \neg d, r$, and $\neg u$. Since no conflict arises and there are still unassigned literals, we step to decision level 3. Suppose we perform a decision setting true the literal s. Fig. 2 show the literals propagated at level 3, namely, $p, q, v, \neg e, e$ and u. Clearly, two conflicts (denoted by the conflict vertices, t_1 and t_2) arise, because of the values to be assigned to u and e, respectively.

The cut corresponding to the first UIP for the conflict t_1 is made of the vertices v, f, and c. The clause learned from this conflict is $\{\neg v, \neg f, \neg c\}$. Similarly, by analyzing the other conflict t_2, we obtain a cut made of the vertices $\neg d$, r, and q, corresponding to the first UIP q, and an alternative learned clause $\{d, \neg r, \neg q\}$..

A solver that decides a single literal in each decision level and that takes into account only one conflict, would learn just one of these two clauses, while both of them can be added to the set of clauses and affect the subsequent part of the execution.

Moreover, performing a global analysis of the implication graph might help in learning alternative clauses. (Note that, the same graph can be obtained by deciding the three literals $\neg b, s, \neg h$ at the first decision level, except for the fact that in that case all vertices belong to level 1.) For instance, the pair of clauses $\{\neg a, \neg p\}$ and $\{\neg g, \neg p\}$ can be obtained by considering the two sets $X_1 = \{a, p\}$ and $X_2 = \{p, g\}$, each of them separating one of the conflicts from all source vertices $\neg b$, $\neg h$, and s. In a situation where one heuristically prefers short clauses, the global analysis produces better results. Observe, moreover, that detecting a single vertex cut separating both conflict vertices from all sources would produce a single learned clause. However, such a clause would contain at least three literals. □

The previous example shows that, even considering simple heuristics based on cardinality of learned clauses (i.e., the number of literals they include), a global analysis may offer advantages. In general, more complex heuristics can be applied to assign weights to literals/vertices and then to vertex cuts. Consequently, the conflict analysis procedure will focus on minumin vertex cuts. Among the various approaches that can be adopted in assigning weights to literals, many of the branching strategies exploited in standard SAT-solvers can be adapted to our case [4].

Remark 1. Note that another generalization can be foreseen. Indeed, it seems natural to consider the parallel execution of several solvers, each one adopting its own criteria and heuristics, and exploring (possibly) different portions of the search space. Plainly, each solver develops a different implication graph. A conjunct analysis of all these graphs (that share the vertices and differ in the set of edges) should be advisable. Intuitively, a common analysis could be enabled by introducing a coloring of the edges. Each color would identify the inferences made by one of the solver. In this way, a global learning process can be developed, by applying global strategies, so as to realize forms of communication and cooperation between the solvers. For the sake of simplicity, in what follows we will not deal with this kind of generalization, which represents an interesting theme for future research.

Summing up, we will consider implication graphs where weights are provided and several conflict vertices may occur. The graphs are still acyclic and layered (as vertices are partitioned by the decision levels). The conflict analysis should focus on those (sets of) cuts that are optimal with respect to a given objective function. Typically, it may encode "local" optimality criteria (focusing on each single conflict in isolation from the others), as well as involve "global" criteria, aimed at optimizing the set of cuts as a whole. To keep the following description as general as possible, we will leave these optimality criteria implicit and simply deal with a generic objective function $f(\cdot)$.

Let $G = (V, E)$ be a weighted DAG with $n = |V|$ vertices and $m = |E|$ edges. Let $S \subseteq V$ be the set of source vertices and $T = \{t_1, \ldots, t_k\} \subseteq (V - S)$ the set of sink vertices. Assume, moreover, that each sink is reachable (through a directed path) from at least one source.

Without loss of generality, we can assume that $S = \{s_1, \ldots, s_k\}$ and that t_i is reachable from s_i (for each i). [3]

A weight function $w : V \to \mathbb{N}$ assigns positive weights to the vertices in G.[4] We are interested in solving the following optimization problem (where $f(\cdot)$ is the objective function).

Problem 1 *Find k vertex sets $X_i \subseteq (V - T)$ that minimize the value of $f(X_1, \ldots, X_k)$ and such that, for each $i \in \{1, \ldots, k\}$, X_i is a vertex cut separating s_i from t_i.*

Example 5. Consider a 'local' optimality criterion that, focusing on single conflict t_i, prefers the vertex cut that produces the smallest learned clause, namely, the one minimizing the cardinality $|X_i|$ of the vertex cut X_i. In presence of k conflicts, two natural choices for the 'global' objective function $f(X_1, \ldots, X_k)$, are:

- $f(X_1, \ldots, X_k) = \sum_i |X_i|$, and
- $f(X_1, \ldots, X_k) = |\bigcup_i X_i|$.

In both cases, the weight function $w(\cdot)$ simply evaluates 1 for each node. Clearly, more complex weight function $w(\cdot)$ and objective functions $f(\cdot)$ can be designed. For instance, $w(\cdot)$ might take into account the number of occurrences of literals in the set of clauses, or the *activity* of literals (cf., [15]), to mention two possibilities. □

A further remark is in order. In what follows we will take advantage from the fact that, any given minimum vertex-cut problem can be translated into a minimum edge-cut problem whose solutions correspond to solutions of the former problem. So, in order to simplify the presentation, in the following sections we formalize our learning scheme in terms of (edge) cut and multicut problems.

In the following sections we will describe some alternative approaches to the solution of Problem 1.

4 Solving the multiple learning problem

Let us start by considering the particular case in which the objective function is the sum of the cuts' weights, i.e., $f(X_1, \ldots, X_k) = \sum_i w(X_i)$. In this case, the problem can

[3] In fact, if this is not the case, we can always find k subsets $S_1, \ldots, S_k$ of S, such that S_i contains all sources from which t_i can be reached, and extend G by adding a new vertex s_i having as successors all vertices in S_i (for each i).

[4] For a set of vertices X let $w(X) = \sum_{x \in X} w(x)$.

be formulated as a *minimum s-t multicut problem* [12] (or dually as a *multicommodity maxflow problem* [19]). Alternatively, one can translate the problem into a *vertex separator problem* on a network, as defined, for instance, in [3]. (The reader is referred to the cited literature, and to [2, 7, 28, 29], for a detailed treatment of these problems.)

A drawback of adopting an encoding in an s-t multicut problem is that the obtained solution would consist in a single vertex/edge set X that acts as a separator for each pair s_i-t_i. This does not fulfill the requirement of Problem 1, that asks for a collection of k cuts (each one separating one of the pairs). Consequently, it is necessary to convert the single cut, by splitting it in k subsets (not necessarily pairwise disjoint). In doing this one has to exploit the s_i-t_i-connectedness of G (for each i) in order to determine which part of X is actually related to the pair s_i-t_i. Computing this step may add a computational cost which is in any case polynomially bounded. (Actually, such an overhead could be absent, if connectedness computations are part of the algorithm used to solve the minimum s-t multicut instance).

Recall that, for general graphs, the problem of finding the minimum s-t multicut is Max SNP-hard [8]. Nevertheless, in undirected graphs [12] proposes an $\mathcal{O}(\log k)$ approximate algorithm based on LP relaxation.

An alternative approach consists in solving Problem 1 through the solution of several minimum s-t cut problems. We outline here two possibilities. The first one consists in independently solving k minimum s-t cut problems, one for each pair s_i-t_i (by ignoring, in each of them, all other pairs s_j-t_j, for $j \neq i$). Clearly, the computational cost of this approach depends on the algorithm exploited to solve each of the k simpler problems. For instance, [14] proposes an algorithm for minimum s-t cut, based on push-relabel methods, having $\mathcal{O}(n \times m \times log(n^2/m))$ time complexity. Notice that, the Hao-Orlin algorithm solves this problem by computing $n-1$ s-t cuts and attains an overall complexity which asymptotically matches that of computing a single s-t cut (cf., [6]). Notice that, in general, the union of the k single minimum s-t cuts does not necessarily represent an optimal solution of the s-t multicut problem. Indeed, it can be easily shown that it is a k-factor approximation of such optimal solution. A similar criticism applies with respect to Problem 1, because finding each single s-t cut independently, does not allow one to impose any global optimization criteria.

The second approach can be adopted when the objective function $f(X_1, \ldots, X_k)$ has an intrinsically global nature, i.e., it is not possible to express it as a simple combination of k simpler functions, each concerning a single pair s_i-t_i (as we did by restraining to the minimization of the sum of the cuts' weights), then a more general technique has to be designed. A viable possibility consists in exploiting some kind of enumeration technique (complete or bounded), such as those proposed in [29, 3, 30], in order to compute one sequence of s_i-t_i-cuts, for each i. Then, the solution to Problem 1 can be obtained by suitably combining k single solutions, each one coming from one of the sequences. More details will be provided in the next section.

5 An optimal solution for the general case

In this section we outline a viable algorithmic approach to Problem 1 that guarantees to achieve the optimal solution. The basic idea consists in enumerating, for each of the k

s_i-t_i pairs, the admissible solutions of the corresponding s_i-t_i cut problem. Then, the optimal solution of the global problem can be sought for by evaluating the objective function $f(X_1, \ldots, X_k)$ directly on the k-tuples of single solutions. In this manner one can carry out the optimization of an objective function in its most general form.

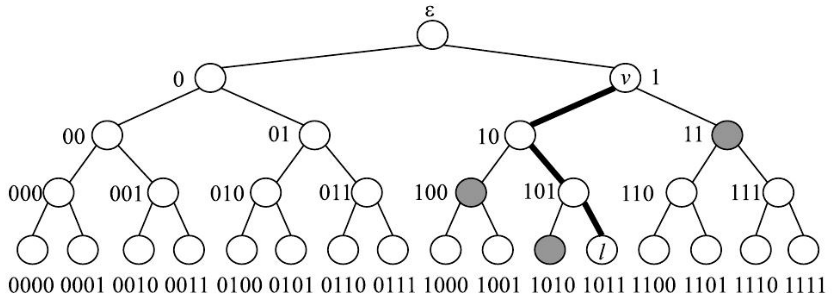
Concerning each single s_i-t_i cut problem, this technique requires to produce an enumeration of all its solutions ordered by non-decreasing weights. Such a specific problem has been formalized precisely in [31] and it turns out to be a $\#P$-complete problem [25].

Concerning the cut problem, instead of the s-t cut problem, a remarkable approach has been proposed in [33], by combining the technique described in [31] with the faster Hao-Orlin algorithm [16]. The resulting algorithm can enumerate all cuts, in non-decreasing order of weight, with a $\mathcal{O}(n \times m \times log(n^2/m))$ delay, between two consecutive solutions.

In our case, we propose the adoption of the same technique (see below), slightly re-engineered in order to obtain an enumeration of all s-t cuts. The resulting naive algorithm is able to solve Problem 1 within $\mathcal{O}(N \times n \times m \times log(n^2/m))$ steps, where N is the number of all s_i-t_i cuts. Notice that, in general, N may be exponential in the size of the given graph [31, 24].

Let us briefly outline the enumeration technique of [33, 31] and how to adapt it to our case. Given a DAG $G(V, E)$, the vertices in V are numbered from 1 to n. In this way, we can represent each cut in G by means of a binary string $b_1, b_2, \ldots, b_n$ composed of n bits, where $b_i = 0$ if and only if the vertex indexed i is in the cut. Conversely, each binary string represents a possible cut.

Consequently, the set of all (strings representing) potential cuts can be organized in a complete binary tree of height n. For example, the picture shown below, borrowed from [33], illustrates such a tree for $n = 4$.



In such a tree, each leaf is associated with one of the possible strings $b_1, b_2, \ldots, b_n$, and *vice versa*. Each internal node, at level h represents a partially specified cut, not involving the vertices indexed from $h + 1$ to n. In other words, such a node of the tree represents the collection of all cuts that agree on the first h vertices. Let this collection be denoted by $C(X, Y)$ with $X = \{i | b_i = 0\}$ and $Y = \{i | b_i = 1\}$. The root of the tree represents all cuts and is denoted by ϵ. For the sake of simplicity, let us identify a node v of the tree with the partially specified cut it represents and let $mc(v)$ denote the minimum cut among those represented by v.

Let $\Pi = \{(\epsilon, mc(\epsilon))\}$ be a working set of pairs where the first component is a node of the tree and the second one is the value of the corresponding minimum cut. The algorithm proceeds by extracting the element $(v, mc(v))$ from Π that minimizes, among the pairs in Π, the value of $mc(v)$. The pair $(v, mc(v))$ is produced as output. Then, the leaf corresponding to the minimum cut $mc(v)$ is considered. Let l be such a cut/leaf (c.f., the above picture). At this point, the path in the tree connecting v to l is considered and for each of the *immediate children* u in such path the pair $(u, mc(u))$ is added to Π. The immediate children are all the nodes which are not in the path, are adjacent to some node in the path, and belong to the tree rooted in v. (They are depicted in gray color in the figure.) Each value $mc(u)$ is obtained by means of an auxiliary computation (for instance, by exploiting the algorithm in [16, 14]). This procedure is repeated until Π is empty.

It is easy to verify that the cuts are enumerated in non-decreasing weight order. Consider once again the above figure. At each step, each of the immediate children u added to Π cannot represent a collection of cuts that includes l. Since l is the optimal solution among all the cuts represented by v, no cut among those represented by u can have weight lower than l. In [31] it is shown that such algorithm, if implemented using suitable data structures, can enumerate all cuts with a $\widetilde{O}(n^2m)$ time delay.[5]

In order to adapt this technique to our purpose, it suffices to consider that to represent s-t cuts we only need a tree of height $n-2$, because s and t must be separated in each cut. Clearly, an auxiliary algorithm will be used to compute minimum s-t cuts, at each step. It might be the case that not all the leaves of the binary tree actually represent s-t cuts. Hence, there might be internal nodes that represent empty collections of s-t cuts. This particular cases are easily dealt with by simply ignoring the corresponding sub-tree. (Notice that, at least one minimum s-t cut exists and it is determined at the beginning of the execution.)

A very inefficient algorithm for Problem 1 is composed of two steps. First, the above outlined enumeration algorithm is used to compute all s_i-t_i cuts $l_1^{(i)}, \ldots, l_{n_i}^{(i)}$, for each $i \in \{1, \ldots, k\}$, where $n_i \leq 2^{n-2}$ is the number of s_i-t_i cuts. Then, the objective function is optimized on the set of k-tuples of the form $l^{(1)}, \ldots, l^{(k)}$, obtaining the set $\overline{l^{(1)}} \cup \cdots \cup \overline{l^{(k)}}$ that minimizes the value of $f(\overline{l^{(1)}}, \ldots, \overline{l^{(k)}})$.

Clearly, the computational complexity of this algorithm is unsatisfactory, even for small graphs. However, one may exploit the specific properties the implication graph exhibits to gain greater efficiency. The structure of the graph $G(V, E)$ has to be considered. Indeed, G is a layered DAG and the application of suitable heuristics might sensibly improve the naive algorithm. For instance, the Padberg-Rinaldi or the Karger heuristics [6] are certainly exploitable. Moreover, the fact that the DAG is an *implication graph* built up by reflecting the logical relationships between propositional variables, implicitly encoded in the set of clauses, has great relevance. In fact, the DAG encodes logical implications between each vertex/literal and the set of its antecedents. By applying simple propositional properties, one can locally re-write the graph G so as to

[5] Actually, one of the crucial points in reducing the overall complexity of the algorithm consists in handling a set Π of pairs $(v, mc(v))$ instead of simple elements v. In this way, in fact, one has to compute the value of $mc(v)$ just once for each node v (see [31] for the details).

transform it into a graph G' which is equivalent to G w.r.t. the learning process, but belongs to a class of DAGs having better computational properties. Examples of graph classes that are desirable targets for this rewriting process are planar graphs and series-parallel graphs. (Note that instead of explicitly applying these rewritings on the graph, one could encode their effect directly in the algorithms user to compute the cuts.)

Another, not antithetic, possibility consists in introducing a bound on the number of non-decreasing cuts to be computed for each s_i-t_i pair. This, in principle, corresponds to accept a solution that approximates the optimal one. To achieve this, the s-t cut algorithm is used to compute the first $c \leq 2^n$ s_i-t_i cuts, for each i, where c is a fixed constant value; Then, the algorithm proceeds as before, but minimizing the objective function w.r.t. the prefixes of the k sequences of s_i-t_i cuts. The quality of the approximation depends on c and on the specific order in which the single s_i-t_i cuts are enumerated by the s_i-t_i cut algorithm.

Further investigation is needed to study all these issues.

6 Concluding remarks and future work

In this paper we considered one of the most crucial component of modern DPLL-based SAT solvers, namely, the conflict analysis procedure.

The purpose of this procedure is the detection of the reasons behind a failure occurred during the search for a satisfying assignment of a SAT instance. By analyzing the failure, caused by a conflicting set of assignments performed by the solver while visiting a solution space, one or more new clauses are learned and added to the problem being solved. The effect of this addition consists in driving the solver "away from the failure", preventing the solver to make again the same contradictory assignments. Such a kind of solving strategy is typical of the so-called conflict-driven SAT-solvers, but similar techniques have been successfully applied in several other fields of computational logic (such as, ASP, CP, SMT, to mention some).

We proposed a formulation of the conflict analysis task in a form expressive enough to reckon with different orthogonal generalizations of the basic schema. Features such as the analysis of multiple conflicts, the generation of multiple learned clauses, the imposition of global optimality criteria, the treatment of multiple decisions, are easily dealt with in the same framework. Extensions to the case of parallel solvers are also foreseeable.

We formalized the general learning problem as the search for a collection of vertex cuts in a directed acyclic graph. Optimality of the solution is evaluated with respect to a given global objective function. Such a function is basically conceived to express (complex) search strategies and heuristics that control the behavior of the solver. Nevertheless, by considering a single conflict and by choosing a simple objective function (e.g., minimizing set cardinality), one can recover the standard conflict analysis schema, normally exploited in common solvers.

We provided ways to face the general learning problem by exploiting well-known algorithms proposed in literature to solve cut and multicut problems on DAGs.

Clearly, the computational effort required to accomplish the learning task in its most general form, is higher than the one needed to learn a collection of single clauses, each of them justifying a single conflict, independently. The advantage of the general technique comes from the potentially higher quality of the learned (sets of) clauses. For instance, the adoption of suitable global objective functions may enable the identification of common reasons for multiple conflicts or the derivation of conflict clauses that involve "heavy" literals, i.e., relevant literals with respect to the heuristics used to determine vertex weights. This may reduce the number of clauses that are learned. Another possibility consists in encoding in the objective-function criteria that tend to minimize the literals shared among the learned clauses. Consequently, a smaller set of clauses would better affect the search, because they would prune different and distant (i.e., loosely related) portions of the search space.

In case the visit of the search space is split in different searches (this can be achieved by assigning from the beginning a set of variables; this corresponds to making a multiple decision at the first decision level) and/or by running different solvers in parallel, the adoption of a global perspective in conflict analysis may help in learning conflict clauses that act as a communication channel between the different searches.

It should be noted that the difference in the computational efforts required by solving the global problem as a whole and solving k simpler problems, reduces whenever one resorts to approximated algorithms. In this context one might benefit even from recent results on fixed-parameter tractability of multicut problems on DAGs [18]. For example, think about the fact that it seems reasonable to fix the number of conflicts to be considered in each analysis (namely, the parameter k) to a predefined value, or to limit the search to those cuts/clauses having a specific cardinality.

In this paper we restrained ourselves to proposing an initial formalization of a general learning schema. We also outlined some simple algorithmic solutions. Much has to be done and there are many challenging themes for future research. As regards the algorithmic aspects, one may explore the applicability to our context of several heuristics and techniques developed for standard multicut problems, such as, for instance, the Padberg-Rinaldi and the Karger heuristics [6].

Introducing a global perspective for multiple learning is certainly interesting *per se*. It might be the case that one benefits from this new perspective in identifying new learning schemata, different from those currently described in literature, especially concerning parallel solvers.

Clearly, the practical advantage of our proposal has to be validated through an extensive experimental activity. In doing this one may proceed by implementing a concrete prototypical solver. Alternatively, one may integrate a general learning schema into an existing solver. The solvers described in [9, 32, 10], are good candidates for this last possibility. In fact, these solvers (as well as those described in [5, 20, 1, 21], to mention some proposals not necessarily concerned with SAT-solving) take advantage from a high-performance parallel architecture, and offer solid support to the implementation of parallel algorithms for conflict analysis.

Acknowledgments

The authors would like to thank Massimo Bernaschi, Alessandro Dal Palù, Agostino Dovier, Giuseppe Italiano, Luca Labellarte, and Enrico Pontelli for suggestions and enlightening discussions on the topics presented in this paper. This work is supported by the GNCS-2014 project.

References

1. A. Arbelaez and P. Codognet. A GPU implementation of parallel constraint-based local search. In *PDP*, pages 648–655. IEEE, 2014.
2. C. Bentz. On the hardness of finding near-optimal multicuts in directed acyclic graphs. *Theor. Comput. Sci.*, 412(39):5325–5332, 2011.
3. A. Berry, J.-P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. In *Graph-Theoretic Concepts in Computer Science*, pages 167–172. Springer, 1999.
4. A. Biere. *Handbook of satisfiability*, vol. 185. Ios PressInc, 2009.
5. F. Campeotto, A. Dal Palù, A. Dovier, F. Fioretto, and E. Pontelli. Exploring the use of GPUs in constraint solving. In M. Flatt and H.-F. Guo, eds., *PADL*, vol. 8324 of *LNCS*, pages 152–167. Springer, 2014.
6. C. S. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 324–333. Society for Industrial and Applied Mathematics, 1997.
7. M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow: a survey. *European Journal of Operational Research*, 162(1):55–69, 2005.
8. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts. In *Proc. of the 24th ACM Symposium on Theory of Computing*, pages 241–251. ACM, 1992.
9. A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. Exploiting unexploited computing resources for computational logics. In *Proc. of the CILC-12*, vol. 857 of *CEUR Workshop Proceedings*, pages 74–88. CEUR-WS.org, 2012.
10. A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. CUD@SAT: SAT solving on GPUs. *Journal of Experimental and Theoretical Artificial Intelligence*, 2014.
11. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
12. N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.
13. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238, 2012.
14. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. of the ACM*, 35(4):921–940, 1988.
15. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. *Discrete Applied Mathematics*, 155(12):1549–1561, 2007.
16. J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proc. of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174. Society for Industrial and Applied Mathematics, 1992.
17. H. Jin and F. Somenzi. Strong conflict analysis for propositional satisfiability. In G. G. E. Gielen, ed., *Proc. of the DATE 2006*, pages 818–823. European Design and Automation Association, Leuven, Belgium, 2006.
18. S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, eds., *Proc. of ICALP*, vol. 7391 of *LNCS*, pages 581–593. Springer, 2012.
19. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. of the ACM*, 46(6):787–832, 1999.
20. P. Manolios and Y. Zhang. Implementing survey propagation on graphics processing units. In A. Biere and C. P. Gomes, eds., *SAT*, vol. 4121 of *LNCS*, pages 311–324. Springer, 2006.
21. R. Marques, L. G. Silva, P. F. Flores, and L. M. Silveira. Improving SAT solver efficiency using a multi-core approach. In C. Boonthum-Denecke and G. M. Youngblood, eds., *FLAIRS Conference*. AAAI Press, 2013.

22. J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
23. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.
24. J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. In V. Rayward-Smith, ed., *Combinatorial Optimization II*, vol. 13 of *Mathematical Programming Studies*, pages 8–16. Springer, 1980.
25. J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.
26. F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.
27. L. Ryan. *Efficient algorithms for clause-learning SAT solvers*. Simon Fraser University, 2004. Master thesis in computer science.
28. H. Shen, K. Li, and S.-Q. Zheng. Separators are as simple as cutsets. In P. S. Thiagarajan and R. H. C. Yap, eds., *Proc. of Advances in Computing Science - ASIAN'99*, vol. 1742 of *LNCS*, pages 347–358. Springer, 1999.
29. H. Shen and W. Liang. Efficient enumeration of all minimal separators in a graph. *Theor. Comput. Sci.*, 180(1-2):169–180, 1997.
30. K. Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discrete Applied Mathematics*, 158(15):1660–1667, 2010.
31. V. V. Vazirani and M. Yannakakis. Suboptimal cuts: Their enumeration, weight and number. In *Automata, Languages and Programming*, pages 366–377. Springer, 1992.
32. F. Vella, A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. CUD@ASP: Experimenting with GPGPUs in ASP solving. In *Proc. of the CILC-12*, vol. 1068 of *CEUR Workshop Proceedings*, pages 163–177. CEUR-WS.org, 2013.
33. L.-P. Yeh, B.-F. Wang, and H.-H. Su. Efficient algorithms for the problems of enumerating cuts by non-decreasing weights. *Algorithmica*, 56(3):297–312, 2010.
34. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in Boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.