Local Lemma: A New Strategy of Pruning in SAT Solvers*

Xiangxue Jia, Runming Lu, Sheng Liu
State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences
{jxx,lurm,lius}@ios.ac.cn
Graduate University, Chinese Academy of Sciences

Jian Zhang
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
zj@ios.ac.cn

ABSTRACT

This paper proposes a search tree pruning strategy for SAT solving. It is called *Local Lemma*, because it generates lemmas from explored subtrees and these lemmas are valid only in a part of the search tree. The paper explains the basic principle of the strategy, illustrates it with an example, and presents some experimental results.

1. INTRODUCTION

Deciding the satisfiability of Boolean formulas (SAT) is a well-known NP-complete problem. Researchers have proposed various efficient methods for solving SAT. Most modern SAT solvers use some kind of learning mechanism to avoid redundant search. See for example [4].

The approach we proposed is motivated by related work on supercubing and B-cubing [1, 3], and our work can be viewed as a compact integration of the B-cubing ideas and the conflict-driven lemma learning mechanism in modern SAT solvers. Like supercubing and B-cubing, the *Local Lemma* strategy is based on this intuition: extracting information in the left branch (also called subtree) of a node, then generating assignments or lemmas from the information and attaching them to the right branch of that node. Specifically, when a conflict u occurs, a decision conflict clause (DCC) is constructed. Its literals are all decision literals. The negation of a DCC records the reason of the conflict, denoted by $Cert(u)$. It is called a **certificate** of unsatisfiability.

2. LOCAL LEMMA

Suppose we check the satisfiability of the formula F. We use chronological backtracking search, and keep recording the search tree. For a node u, let $D(u)$ denote the conjunction of the decision literals from root to u. If S is a set of nodes, define $D(S) = \{D(u)|u \in S\}$.

Specially, if u is a leaf node, it means that the procedure finds a solution or ends up with a conflict. If u stands for a deadend, we can conclude that the formula $F \wedge D(u)$ has no solution. In order to consider the influence of a decision x, without loss of generality, suppose $D(u) = Q \wedge x$.

DEFINITION 2.1. *Suppose the search procedure reaches a conflict in the node u, and $D(u) = Q \wedge x$ where x is a decision variable. Decision x is said to be independent of the conflict if $F \wedge Q$ is unsatisfiable.*

When the search meets a conflict, we can use DCC analysis to obtain a DCC($\neg Cert(u)$) whose variables are all decision variables. Recall that $D(u) = Q \wedge x$ is the conjunction of all decision literals in the conflicting stage. So $Q \wedge x \implies Cert(u)$. If $Cert(u)$ does not contain x, the formula can be reduced to

$$Q \implies Cert(u) \tag{2.1}$$

And $\neg Cert(u)$ is a learning clause, so

$$F \implies \neg Cert(u) \tag{2.2}$$

Thus,

$$F \wedge Q \implies FALSE \tag{2.3}$$

So we conclude x is independent of the conflict if $Cert(u)$ doesn't contain x.

It's easy to see if x is independent of the conflict $F \wedge Q \wedge x$, the formula $F \wedge Q \wedge \neg x$ is also unsatisfiable. In this situation, we immediately obtain the knowledge that there is no solution even we simply flip x, so we need not visit the search space denoted by $F \wedge Q \wedge \neg x$. In this way, we can prune some search space by utilizing the concept of independence. From the view of Supercubing and B-cubing, this nature is a kind of symmetry; From our point of view, it is a kind of independence. In order to make full use of the concept of independence to prune the search space, in the following we first formalize some notations of the search procedure.

DEFINITION 2.2. *Suppose $a_i(i = 1, 2, \ldots, m)$ is a conjunction of literals. If the disjunction of $a_i(i = 1, 2, \ldots, m)$ is a tautology and for any two a_i and a_j $(i \neq j)$ $a_i \wedge a_j$ is not satisfiable, then the set of $a_i(i = 1, 2, \ldots, m)$ is said to be a partition of search space.*

THEOREM 2.3. *Suppose a chronological backtracking procedure generates a search tree T. Let S be the set of leaf nodes of T. Then $D(S)$ is a partition of search space.*

Consider a subtree T' which stands for the procedure exploring the formula $F' \wedge x$. This branch has been explored sufficiently and no solution was found. Let S be the set of leaf nodes of this branch.

*Supported by the National Natural Science Foundation of China (Grant No. 60673044).

SAC'10 March 22-26, 2010, Sierre, Switzerland.

From Theorem 2.3, we know $D(S)$ is a partition of search space. Suppose $D(S) = \{a_1, a_2, ..., a_n\}$. So,

$$a_1 \vee a_2 \vee ... \vee a_n \Leftrightarrow true$$

$$F' \wedge x \Leftrightarrow (F' \wedge x \wedge a_1) \vee (F' \wedge x \wedge a_2) \vee ... \vee (F' \wedge x \wedge a_n)$$

We know that each leaf node stands for a formula $F' \wedge x \wedge a_i$ and each leaf node is a deadend. In some deadends, the decision x is independent of the conflict of the formula $F' \wedge x \wedge a_i$, which means for some i, $F' \wedge a_i$ is $obviously$ unsatisfiable. When exploring the branch of $F' \wedge \neg x$,

$$F' \wedge \neg x \Leftrightarrow (F' \wedge \neg x \wedge a_1) \vee (F' \wedge \neg x \wedge a_2) \vee ... \vee (F' \wedge \neg x \wedge a_n)$$

After deleting all the $obviously$ unsatisfiable $F' \wedge a_i$, we obtain

$$F' \wedge \neg x \Leftrightarrow (F' \wedge \neg x \wedge a_{j_1}) \vee (F' \wedge \neg x \wedge a_{j_2}) \vee ... \vee (F' \wedge \neg x \wedge a_{j_m})$$

$$F' \wedge \neg x \implies a_{j_1} \vee a_{j_2} \vee ... \vee a_{j_m} \tag{2.4}$$

So, we can extract lemmas from

$$a_{j_1} \vee a_{j_2} \vee ... \vee a_{j_m} \tag{2.5}$$

These lemmas are valid only in the branch of exploring the formula $F' \wedge \neg x$, so they are called *local lemmas*.

The following example is modified from [1]. See the left tree in Figure 1. In the process of exploring subtree d, the solver obtains the following certificates:

$Cert(e) = x_2 \wedge x_4$
$Cert(m) = \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge x_6$
$Cert(n) = \neg x_1 \wedge \neg x_5 \wedge \neg x_6$
$Cert(j) = x_2 \wedge x_7$
$Cert(k) = x_2 \wedge x_5 \wedge x_6$
$Cert(l) = \neg x_1 \wedge \neg x_3 \wedge \neg x_4 \wedge x_5 \wedge \neg x_6$

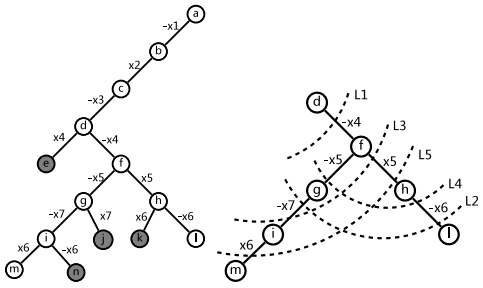

Figure 1: The search tree and the cuts

From these certificates, we know that decision $\neg x_3$ is independent of the conflicts of node e, n, j, k. According to formula 2.5, in the right subtree of c, ignoring all the decisions before c, we can extract local lemmas from the formula $D(m) \vee D(l)$ which is:

$$(\neg x_4 \wedge \neg x_5 \wedge \neg x_7 \wedge x_6) \vee (\neg x_4 \wedge x_5 \wedge \neg x_6) \tag{2.6}$$

$$\Leftrightarrow \neg x_4 \wedge (\neg x_5 \vee \neg x_6) \wedge (\neg x_7 \vee x_5) \wedge (\neg x_7 \vee \neg x_6) \wedge (x_6 \vee x_5)$$

So before we begin to explore x_3, we can record the following local lemmas: $\neg x_4$, $\neg x_5 \vee \neg x_6$, $\neg x_7 \vee x_5$, $\neg x_7 \vee \neg x_6$, $x_6 \vee x_5$.

As mentioned above, local lemmas are extracted from formula 2.5. In the corresponding subtree of the search procedure, we can map each edge to the corresponding decision literal, and a_{j_i} can be viewed as a path from the root to the corresponding leaf node. So formula 2.5 can be viewed as a disjunction of all paths of a binary tree. If it's transformed to CNF, a clause corresponds to a cut through all paths. So our problem is how to find cuts in a binary tree. We use the divide-and-conquer strategy. Define $C(T)$ as all cuts of binary tree T. Let T_L and T_R be children of T. A cut of T must consist of a cut of T_L and a cut of T_R. So $C(T) = C(T_L) \times C(T_R)$.

At first glance, this procedure may obtain too many local lemmas. But we have several strategies to prevent exponential explosion: First, some clauses(e.g. tautologies) can be eliminated. Second, long clauses do not need to be recorded. Third, we set an upper bound on the number of local lemmas.

Take formula 2.6 for an example. It corresponds to a subtree of the tree on the left side of Figure 1. There are 5 cuts in the tree(except for the cuts corresponding to tautologies). See the right tree in Figure 1: $L_1 = \neg x_4$, $L_2 = \neg x_7 \vee \neg x_6$, $L_3 = \neg x_7 \vee x_5$, $L_4 = \neg x_5 \vee \neg x_6$, $L_5 = x_6 \vee x_5$, the same as the local lemmas computed by logical deduction on formula 2.6.

3. EXPERIMENTAL RESULTS

Based on the above ideas, we implemented a tool, called *LocalLemma*, on top of MiniSAT2.0 [2]. We tested the tool on a number of well-known problems and compared it with MiniSAT2.0. The experimental results are shown in Table 1.

Table 1. MiniSAT V.S. *LocalLemma*

	MiniSAT			*LocalLemma* with VSIDS		
benchmark set	Time	Decisions	Conflict	Time	Decisions	Conflict
IBM BMC(13)	13.8	147444	38050	35.5	111855	94300
CMU BMC(34)	639	6231625	1329565	522	1494034	1326117
Int Fact(29)	5054	19790750	16892158	6251	17007925	16982421
Frb30-Frb40(15)	1678	2868211	1941176	2813	2011053	3016927
QG(22)	22.8	177800	146995	31.7	151783	149102

The results in Table 1 show that *LocalLemma* is comparable with MiniSAT. And it makes fewer decisions. For *LocalLemma*, the ratio of conflicts to decisions is bigger than MiniSAT. Generally speaking, compared with MiniSAT, each decision of *LocalLemma* can induce relatively more conflicts. This shows the effectiveness of *LocalLemma* in finding conflicts during the search process.

4. CONCLUSIONS AND FUTURE WORKS

Local Lemma is a new strategy of pruning in SAT solver. It prunes more and makes fewer decisions. At the same time, the ratio of conflicts to decisions is bigger. This means *Local Lemma* can find conflicts more easily.

To keep the binary trace tree, we use chronologic backtracking. But supercubing and B-cubing use nonchronological backtracking, which has certain advantages. We leave the combination of *Local Lemma* and nonchronological backtracking as future work.

5. REFERENCES

[1] D. Babic, J. Bingham, and A. J. Hu. Efficient SAT solving: Beyond supercubes. In *Proc. of the 42nd Design Automation Conference*, pages 744–749, 2005.

[2] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. of SAT'03*, pages 502–518, 2003.

[3] E. Goldberg, M. R. Prasad, and R. K. Brayton. Using problem symmetry in search based satisfiability algorithms. In *Proc. of the Conference on Design, Automation, and Test in Europe*, pages 134–142, 2002.

[4] J.P. Marques-Silva and K.A. Sakallah. GRASP - a new search algorithm for satisfiability. In *IEEE International Conf. on Tools with Artificial Intelligence*, 1996.