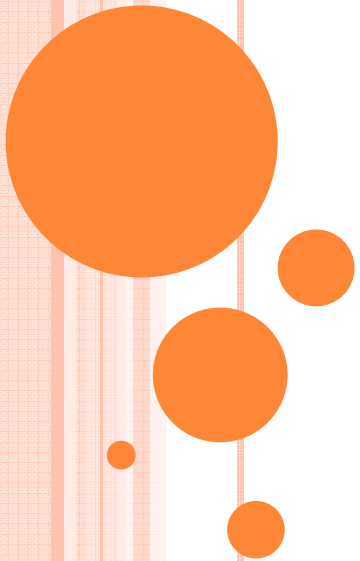


Modern techniques for solving Boolean satisfiability

GRASP

- Learn from the mistake that led to the conflict and introduces additional implicates to the clause database only when it stumbles.
- Conflict diagnosis produces three distinct pieces of information that can help speed up the search:

1. New implicates that did not exist in the clause database and that can be identified with the occurrence of the conflict. These clauses may be added to the clause database to avert future occurrence of the same conflict and represent a form conflict-based equivalence (CBE).

2. An indication of whether the conflict was ultimately due to the most recent decision assignment or to an earlier decision assignment.

a) If that assignment was the most recent (i.e. at the current decision level), the opposite assignment (if it has not been tried) is immediately implied as a necessary consequence of the conflict; we refer to this as a *failure-driven assertion* (FDA).

b) If the conflict resulted from an earlier decision assignment (at a lower decision level), the search can backtrack to the corresponding level in the decision tree since the subtree rooted at that level corresponds to assignments that will yield the same conflict. The ability to identify a backtracking level that is much earlier than the current decision level is a form of non-chronological backtracking that we refer to as *conflict-directed backtracking* (CDB), and has the potential of significantly reducing the amount of search.

- Let the assignment of a variable x be implied due to a clause $w = (l_1 + \ldots + l_k)$. The *antecedent assignment* of x, denoted as $A(x)$, is defined as the set of assignments to variables other than x with literals in w.
- Intuitively, $A(x)$ designates those variable assignments that are directly responsible for implying the assignment of x due to w. For example, the antecedent assignments of x, y and z due to the clause $w = (x + y + z)$ are, respectively:
  - $A(x) = \{y=0, z=1\}$
  - $A(y) = \{x=0, z=1\}$
  - $A(z) = \{x=0, y=0\}$
- Note that the antecedent assignment of a decision variable is empty.

- The sequence of implications generated by BCP is captured by a directed implication graph I defined as follows:
  - Each vertex in I corresponds to a variable assignment $x = v(x)$
  - The predecessors of vertex x=v(x) in I are the antecedent assignments A(x) corresponding to the unit clause w that led to the implication of x. The directed edges from the vertices in A(x) to vertex x=v(x) are all labeled with w. Vertices that have no predecessors correspond to decision assignments.
  - Special conflict vertices k are added to I to indicate the occurrence of conflicts. The predecessors of a conflict
  - The decision level of an implied variable x is related to those of its antecedent variables according to:

$$\delta(x) = \max \{\delta(y) | (y, v(y)) \in A(x)\}$$

Clause database and partial implication graph

Current Assignment:

$$\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2\}$$

Decision Assignment:

$$\{x_1 = 1@6\}$$

$$\omega_1 = (\neg x_1 + x_2)$$

$$\omega_2 = (\neg x_1 + x_3 + x_9)$$

$$\omega_3 = (\neg x_2 + \neg x_3 + x_4)$$

$$\omega_4 = (\neg x_4 + x_5 + x_{10})$$

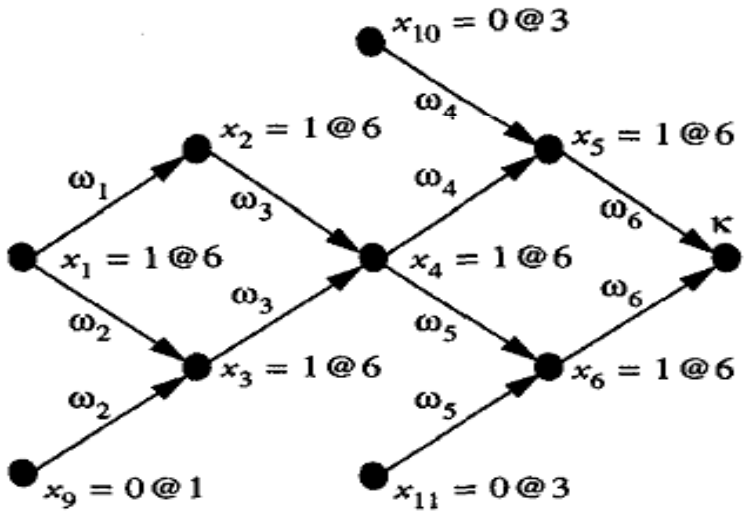
$$\omega_5 = (\neg x_4 + x_6 + x_{11})$$

$$\omega_6 = (\neg x_5 + \neg x_6)$$

$$\omega_7 = (x_1 + x_7 + \neg x_{12})$$

$$\omega_8 = (x_1 + x_8)$$

$$\omega_9 = (\neg x_7 + \neg x_8 + \neg x_{13})$$



Clause Database — Implication Graph

- Figura 1

Conflict Analysis Procedure

- When a conflict arises during BCP, the structure of the implication sequence converging on a conflict vertex k is analyzed to determine those (unsatisfying) variable assignments that are directly responsible for the conflict. The conjunction of these conflicting assignments is an implicant that represents a sufficient condition for the conflict to arise. Negation of this implicant, therefore, yields an implicate of the Boolean function f (whose satisfiability we seek) that does no exists in the clause database φ. This new implicate is referred to as a *conflict-induced clause*.

- We denote the conflicting assignment associated with a conflict vertex k by $A_C(k)$ and the associated conflict-induced clause by $w_c(k)$.

- We partition $A_C(k)$ in two sets:

$$\Lambda(x) = \{ (y, \nu(y)) \in A(x) | \delta(y) < \delta(x) \}$$
$$\Sigma(x) = \{ (y, \nu(y)) \in A(x) | \delta(y) = \delta(x) \}$$

Es. Figure 1 : $\Lambda(x_6) = \{x_{11}=0@3\}$ and $\Sigma(x_6) = \{x_4=1@6\}$.

- Determination of the conflict assignment $A_C(k)$ can now be computed using the following definition:

$$A_C(x) = \begin{cases} (x, \nu(x)) & \text{if } A(x) = \varnothing \\ \Lambda(x) \cup \left[\bigcup_{(y, \nu(y)) \in \Sigma(x)} A_C(y) \right] & \text{otherwise} \end{cases} \quad (3)$$

- starting with x=k. The conflict-induced clause corresponding to is now determined according to:

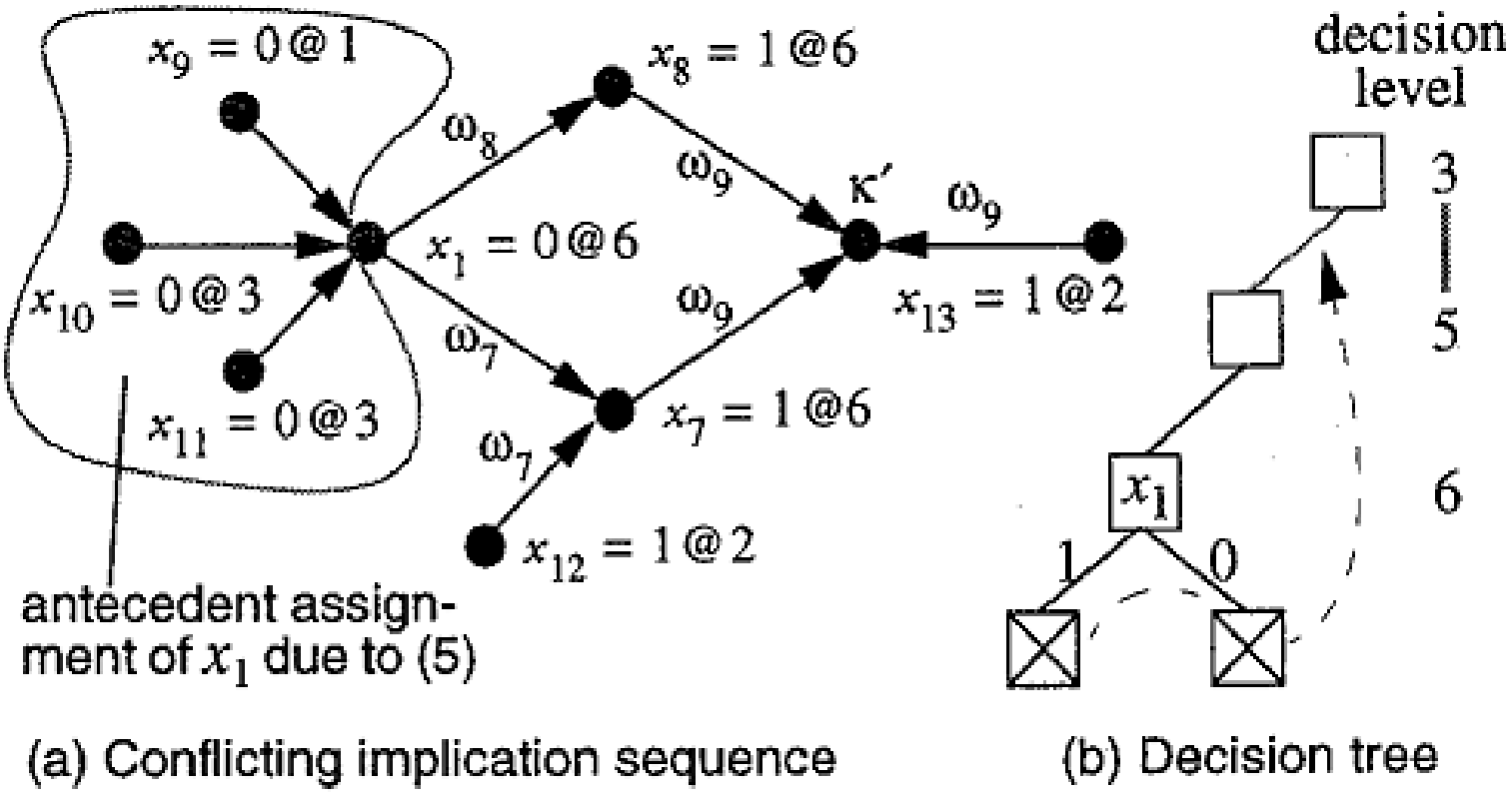
$$\omega_C(\kappa) = \sum_{(x, \nu(x)) \in A_C(\kappa)} x^{\nu(x)} \quad (4)$$

where, for a binary variable x, $x^0 \equiv x$ $x^1 \equiv \underline{x}$.

- Es. on fig. 1:

$$A_C(\kappa) = \{x_1 = 1, x_9 = 0, x_{10} = 0, x_{11} = 0\}$$
$$\omega_C(\kappa) = (\neg x_1 + x_9 + x_{10} + x_{11}) \quad (5)$$

NON- CHRONOLOGICAL BACKTRACKING



- Figura 3

FAILURE-DRIVEN ASSERTIONS:

- If $w_c(K)$ involves the current decision variable, erasing the implication sequence at the current decision level makes $w_c(k)$ a unit clause and causes the immediate implication of the decision variable to its opposite value. We refer to such assignments as failure- driven assertions (FDAs).
- Using our running example (Fig. 1) as an illustration, we note that after erasing the conflicting implication sequence at level 6, the conflict-induced clause $w_c(k)$ in (5) becomes a unit clause with $\underline{x}_1$ as its free literal. This immediately implies the assignment $x_1=0$ and x_1 is said to be asserted.

CONFLICT-DIRECTED BACKTRACKING:

- If all the literals in $w_c(K)$ correspond to variables that were assigned at decision levels that are lower than the current decision level, we can immediately conclude that the search process needs to backtrack. This situation can only take place when the conflict in question is produced as a direct consequence of diagnosing a previous conflict and is illustrated in Figure 3 for our working example. The implication sequence generated after asserting $x_1=0$ due to conflict k leads to another conflict k'.
- The conflicting assignment and conflict-induced clause associated with this new conflict are easily determined to be:

$$A_C(\kappa') = \{x_9 = 0, x_{10} = 0, x_{11} = 0, x_{12} = 1, x_{13} = 1\} \quad (6)$$

$$\omega_C(\kappa') = (x_9 + x_{10} + x_{11} + \neg x_{12} + \neg x_{13})$$

- and clearly show that the assignments that led to this second conflict were all made prior to the current decision level.

- In such cases, it is easy to show that no satisfying assignments can be found until the search process backtracks to the highest decision level at which assignments in $A_C(k')$ were made. Denoting this backtrack level by β, it is simply calculated according to:

$$\beta = \max \{ \delta(x) | (x, v(x)) \in A_C(\kappa') \} \qquad (7)$$

- When $\beta = d-1$, where d is the current decision level, the search process backtracks **chronologically** to the immediately preceding decision level. When $\beta < d-1$, however, the search process may backtrack **non-chronologically** by jumping back over several levels in the decision tree.
- It is worth noting that all truth assignments that are made after decision level β will force the just-identified conflict-induced clause $w_C(k')$ to be unsatisfied.
- For our example, after occurrence of the second conflict the backtrack decision level is calculated, from (7), to be 3. Backtracking to decision level 3, the deduction engine creates a conflict vertex corresponding to $w_C(k')$.

zChaff
Unit Clause Rule (1/2)

- A clause is implied iif all but one of its literals is assigned to zero.
- So, to implement BCP efficiently, we wish to find a way to quickly visit newly implied all clauses that become *newly implied* by a single addition to a set of assignments.
- If the clause has N literals, there is really no reason that we need to visit it when 1, 2, 3, 4, … , N-1 literals are set to zero. We would like to only visit it when the "number of zero literals" counter goes from N-2 to N-1. As an approximation to this goal, we can pick any two literals not assigned to 0 in each clause to watch at any given time. Thus, we can guarantee that until one of those two literals is assigned to 0, there cannot be more than N-2 literals in the clause assigned to zero, that is, the clause is not implied. Now, we need only visit each clause when one of its two *watched literals* is assigned to zero.

zChaff
Unit Clause Rule (2/2)

- When we visit each clause, one of two conditions must hold:

1. The clause is not implied, and thus at least 2 literals are not assigned to zero, including the other currently watched literal. This means at least one non-watched literal is not assigned to zero. We choose this literal to replace the one just assigned to zero. Thus, we maintain the property that the two watched literals are not assigned to 0.
2. The clause is implied. Follow the procedure for visiting an implied clause. One should take note that the implied variable must always be the other watched literal, since, by definition, the clause only has one literal not assigned to zero, and one of the two watched literals is now assigned to zero.

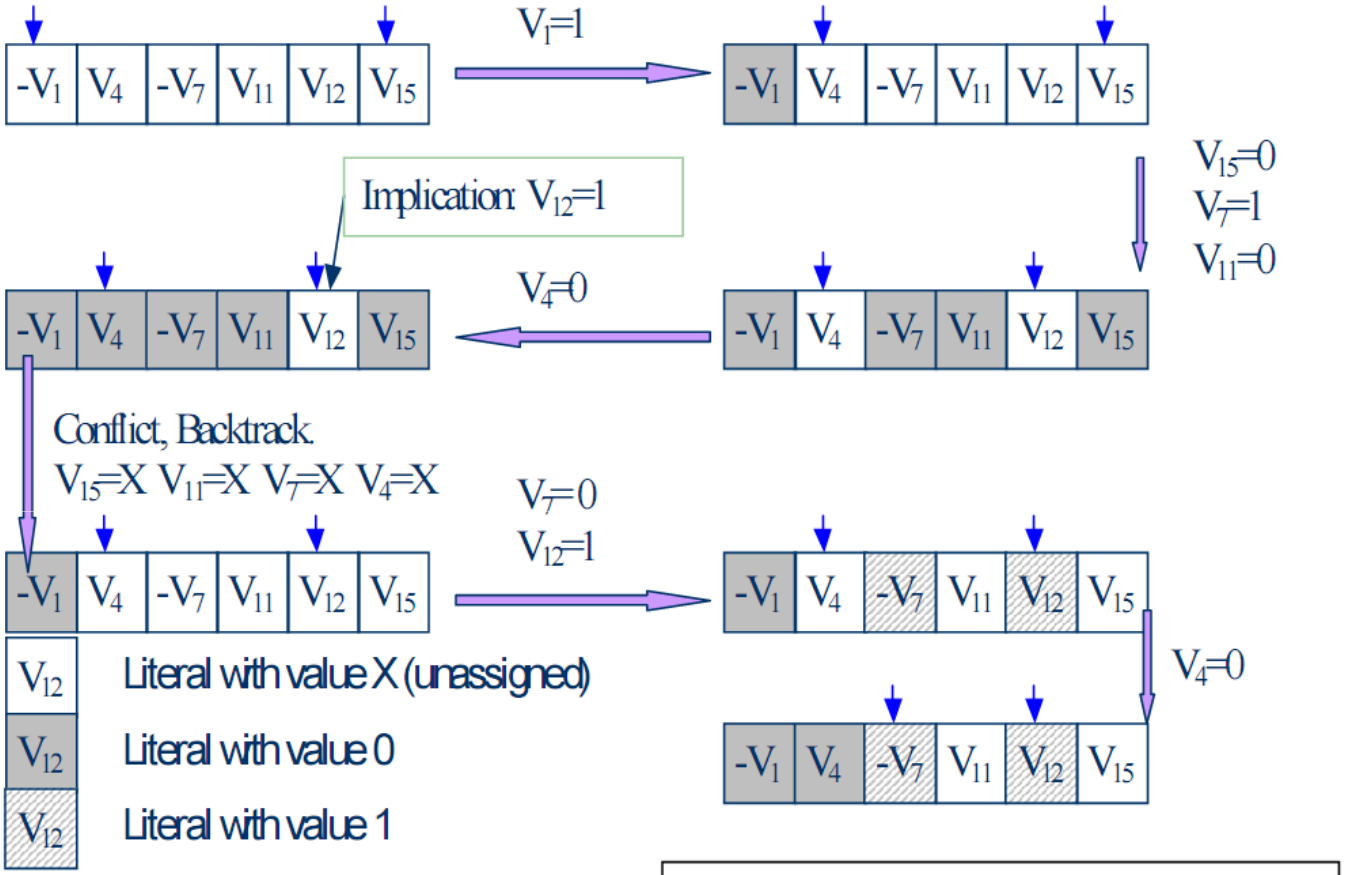


Figure 1: BCP using two watched literals

DECISION HEURISTICS:

- Decision assignment consists of the determination of which new variable and state should be selected each time a decision has to be made. A lack of clear statistical evidence supporting one decision strategy over others has made it difficult to determine what makes a good decision strategy and what makes a bad one. We have many strategies available and so it is important to understand how best to evaluate them.
- One possible metrics:
  - Consider, for instance, the number of decisions performed by the solver when processing a given problem. Since this statistic has the feel of a good metric for analyzing decision strategies ought to mean smarter decisions were made, the reasoning goes – it has been used almost exclusively as the comparator in the scant literature on the subject.

But…

- …not all decisions yield an equal number of BCP operations!
- As a result, a shorter sequence of decisions may actually lead to more BCP operations than a longer sequence of decisions, begging the question:
- What does the number of decisions really tell us?
- The same argument applies to statistics involving conflicts.
- Furthermore, it is also important to recognize that not all decision strategies have the same computational overhead, and as a result, the "best" decision strategy combination of the available computation statistics actually be the slowest if the overhead is significant enough.

CONCLUSION:

- All we really want to know is which strategy is fastest, regardless of the computation statistics.

- Strategy of ZCHAFF can be viewed as attempting to satisfy recent the conflict clauses but particularly attempting to satisfy *recent* conflict clauses.

OTHER FEATURES

- Like many other solvers, Chaff supports the deletion of *added* conflict clauses added to avoid a memory explosion.

- Chaff also employs a feature referred to as restarts. Restarts in general consist of a halt in the solution process, and a restart of the analysis, with some of the information gained from the previous analysis included in the new one. As implemented in **zChaff**, a restart consists of clearing the state of all the variables (including all the decisions) then proceeding as normal.

MINISAT

```
loop
    propagate()     – propagate unit clauses
    if not conflict then
        if all variables assigned then
            return SATISFIABLE
        else
            decide()     – pick a new variable and assign it
    else
        analyze()     – analyze conflict and add a conflict clause
        if top-level conflict found then
            return UNSATISFIABLE
        else
            backtrack()     – undo assignments until conflict clause is unit
```

MINISAT FEATURES

- Unit Propagation with watched literals
- Learning procedure derived by GRASP. The number of learnt clauses are periodically reduced in order to avoid memory explosion.
- Non-chronological backtracking
- Activity heuristic:

1. Bumping: every time a variable occurs in a recorded conflict clause, its activity is increased.
2. Decaying: after recording a conflict, the activity of all the variables in the system are multiplied by a constant less than 1.

- Restarts.