Boolean Satisfiability

ICS 275
Spring 2010

Learning Issues

- Learning styles
  - Graph-based or context-based
  - i-bounded, scope-bounded
  - Relevance-based
- Non-systematic randomized learning
- Implies time and space overhead
- Applicable to SAT

Boolean Satisfiability & Optimization

J. Marques-Silva[1]

[1]University College Dublin, Ireland

ECAI 2010

Part I

Definitions, SAT Algorithms & Modelling Techniques

Outline

Preliminaries

Algorithms

SAT-Based Modelling

Basic Definitions

- Propositional variables can be assigned value 0 or 1
  - In some contexts variables may be unassigned
- A clause is satisfied if at least one of its literals is assigned value 1

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unsatisfied if all of its literals are assigned value 0

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unit if it contains one single unassigned literal and all other literals are assigned value 0

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A formula is satisfied if all of its clauses are satisfied
- A formula is unsatisfied if at least one of its clauses is unsatisfied

Pure Literals

- A literal is pure if only occurs as a positive literal or as a negative literal in a CNF formula
  - Example:
    $\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
  - x_1 and x_3 and pure literals
- Pure literal rule:
  Clauses containing pure literals can be removed from the formula
  (i.e. just assign pure literals to the values that satisfy the clauses)
  - For the example above, the resulting formula becomes:
    $\varphi = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
- A reference technique until the mid 90s; nowadays seldom used

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

- Unit propagation can satisfy clauses but can also unsatisfy clauses (i.e. conflicts)

Resolution

- Resolution rule:
  - If a formula φ contains clauses $(x \vee \alpha)$ and $(\neg x \vee \beta)$, then infer $(\alpha \vee \beta)$

$$\mathrm{RES}(x \vee \alpha, \neg x \vee \beta) = (\alpha \vee \beta)$$

- Resolution forms the basis of a complete algorithm for SAT
  - Iteratively apply the following steps: [Davis&Putnam, JACM'60]
    - Select variable x
    - Apply resolution rule between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$
    - Remove all clauses containing either x or $\neg x$
    - Apply the pure literal rule and unit propagation
  - Terminate when either the empty clause or the empty formula is derived

Resolution – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

Resolution – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

Resolution – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

Resolution – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

Resolution – An Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

(x_3)

- Formula is SAT

Outline

Preliminaries

Algorithms

- Local Search
- The DPLL Algorithm
- Conflict-Driven Clause Learning (CDCL)

SAT-Based Modelling

Algorithms for SAT

- Incomplete algorithms (i.e. cannot prove unsatisfiability):
  - Local search / hill-climbing
  - Genetic algorithms
  - Simulated annealing
  - ...
- Complete algorithms (i.e. can prove unsatisfiability):
  - Proof system(s)
    - Natural deduction [e.g. Huth & Ryan'04]
    - Resolution
    - Stalmarck's method
    - Recursive learning
    - ...
  - Binary Decision Diagrams (BDDs)
  - Backtrack search / DPLL
    - Conflict-Driven Clause Learning (CDCL)
  - ...

Outline

DPLL – Historical Perspective

- In 1960, M. Davis and H. Putnam proposed the DP algorithm:
  - Resolution used to eliminate 1 variable at each step
  - Applied the pure literal rule and unit propagation
- Original algorithm was inefficient
- In 1962, M. Davis, G. Logemann and D. Loveland proposed an alternative algorithm:
  - Instead of eliminating variables, the algorithm would split on a given variable at each step
  - Also applied the pure literal rule and unit propagation
- The 1962 algorithm is actually an implementation of backtrack search
- Over the years the 1962 algorithm became known as the DPLL (sometimes DLL) algorithm

The DPLL Algorithm

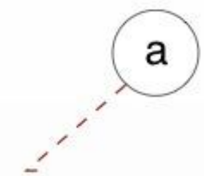
- Standard backtrack search
- At each step:
  - [DECIDE] Select decision assignment
  - [DEDUCE] Apply unit propagation and (optionally) the pure literal rule
  - [DIAGNOSE] If conflict identified, then backtrack
    - If cannot backtrack further, return UNSAT
    - Otherwise, proceed with unit propagation
  - If formula satisfied, return SAT
  - Otherwise, proceed with another decision

An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$

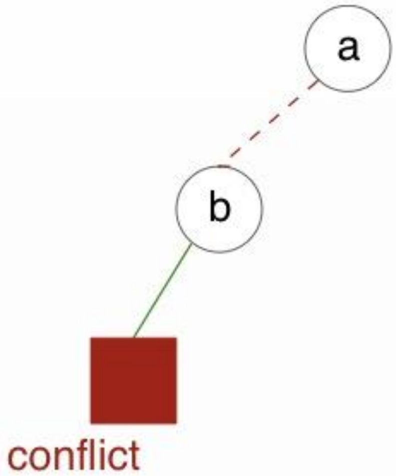
An Example of DPLL

$$
\begin{aligned}
\varphi \;=\;& (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



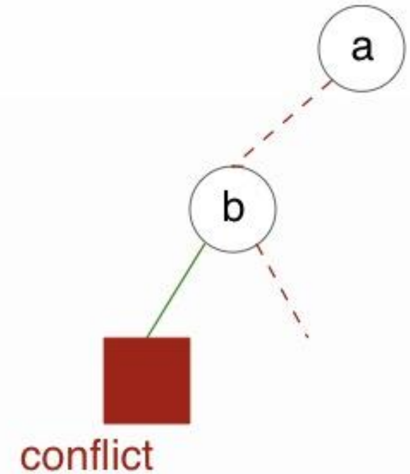
An Example of DPLL

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



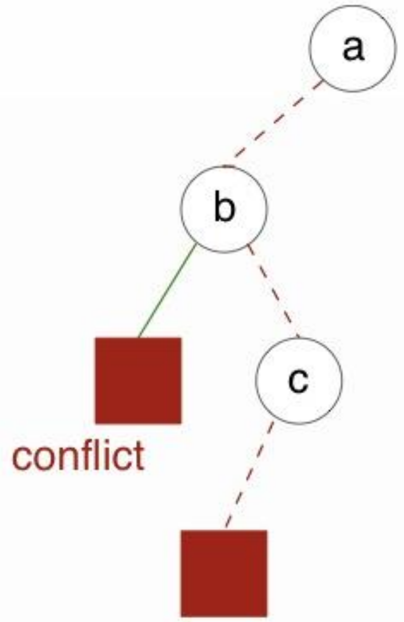
An Example of DPLL

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



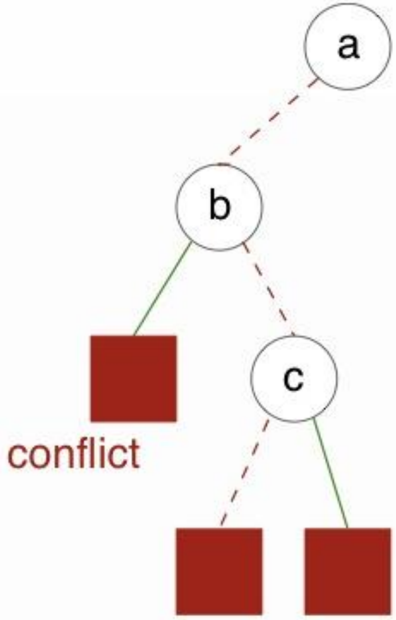
An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



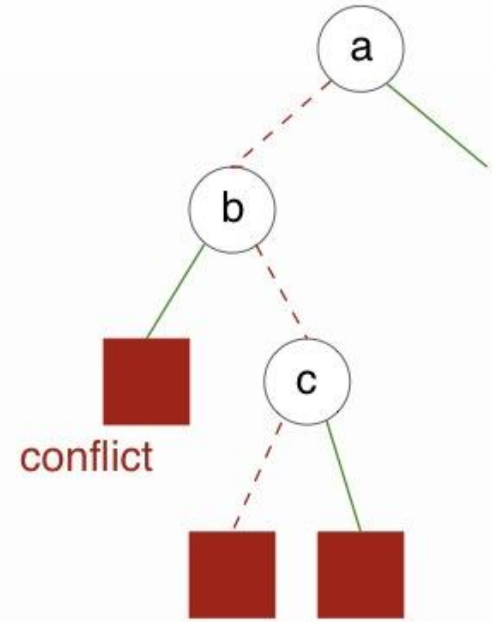
An Example of DPLL

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



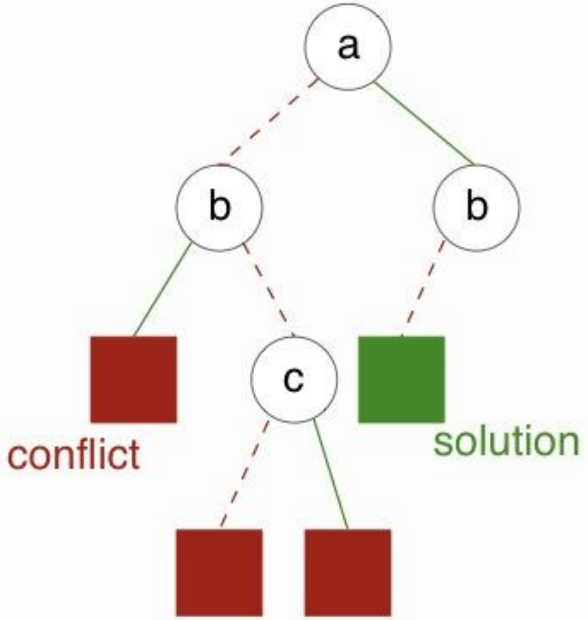
An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



An Example of DPLL

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



Comparing with CSP:

- Sat can be decided before all variables are assigned

Complexity: when is unit propagation complete?....

Think Horn clauses

Outline

CDCL SAT Solvers – Basic Techniques

- Based on DPLL [Davis et al., JACM'60, CACM'62]
  - Must be able to prove unsatisfiability
- New clauses are learned from conflicts [Marques-Silva&Sakallah, ICCAD'96]
  - Backtracking can be non-chronological
- Structure of conflicts is exploited (UIPs) [Marques-Silva&Sakallah, ICCAD'96]
- Backtrack search is periodically restarted [Gomes et al., AAAI'98]
- Lazy data structures are used [Moskewicz et al, DAC'01]
  - Compact with low maintenance overhead
- Branching is guided by conflicts [Moskewicz et al, DAC'01]
  - E.g. VSIDS, etc.

CDCL SAT Solvers – Additional Techniques

- (Currently) effective techniques:
  - Unused learned clauses are discarded [Goldberg&Novikov, DATE'02]
  - Use formula preprocessing I [Een&Biere, SAT'05]
  - Minimize learned clauses [Sorensson&Biere, SAT'09]
  - Use literal progress saving [Pipatsrisawat&Darwiche, SAT'07]
  - Use dynamic restart policies [Biere, SAT'08]
  - Exploit extended implication graphs [Audemard et al., SAT'08]
  - Identify glue clauses [Audemard & Simon, IJCAI'09]

- (Currently) ineffective techniques:
  - Identify pure literals [Davis&Putnam, JACM'60]
  - Implement variable lookahead [Anbulagan&Li, IJCAI'97]
  - Use formula preprocessing II [Brafman, IJCAI'01]

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation

- Unit clause rule: [Davis&Putnam, JACM'60]
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

- Unit propagation can satisfy clauses but can also unsatisfy clauses (i.e. conflicts)

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (\boxed{a} \vee b) \wedge (\neg b \vee \boxed{c} \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee \boxed{f}) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (\boxed{a} \vee b) \wedge (\neg b \vee \boxed{c} \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee \boxed{f}) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (\boxed{a} \vee b) \wedge (\neg b \vee \boxed{c} \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee \boxed{f}) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$
  - Learn new clause $(a \vee c \vee f)$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\begin{aligned}\varphi \; = \; & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$
\begin{aligned}
\varphi \;=\; & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\
& (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)
\end{aligned}
$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\begin{aligned}\varphi \;=\;& (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land \\ & (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)\end{aligned}$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \lor c \lor f)$ implies $a = 1$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$
\begin{aligned}
\varphi \;=\; & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\
& (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)
\end{aligned}
$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\begin{aligned}\varphi \;=\; & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\begin{aligned}\varphi \;=\; & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\begin{aligned}\varphi \;=\; & (a \lor b) \land (\neg b \lor \boxed{c} \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor \boxed{f}) \land \\ & (a \lor \boxed{c} \lor \boxed{f}) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)\end{aligned}$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \lor c \lor f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \lor \neg e \lor f)$ is unsatisfied
  - $(c = 0) \land (f = 0) \Rightarrow (\varphi = 0)$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\begin{aligned}\varphi \;=\; & (a \vee b) \wedge (\neg b \vee \boxed{c} \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee \boxed{f}) \wedge \\ & (a \vee \boxed{c} \vee \boxed{f}) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (c = 1) \vee (f = 1)$

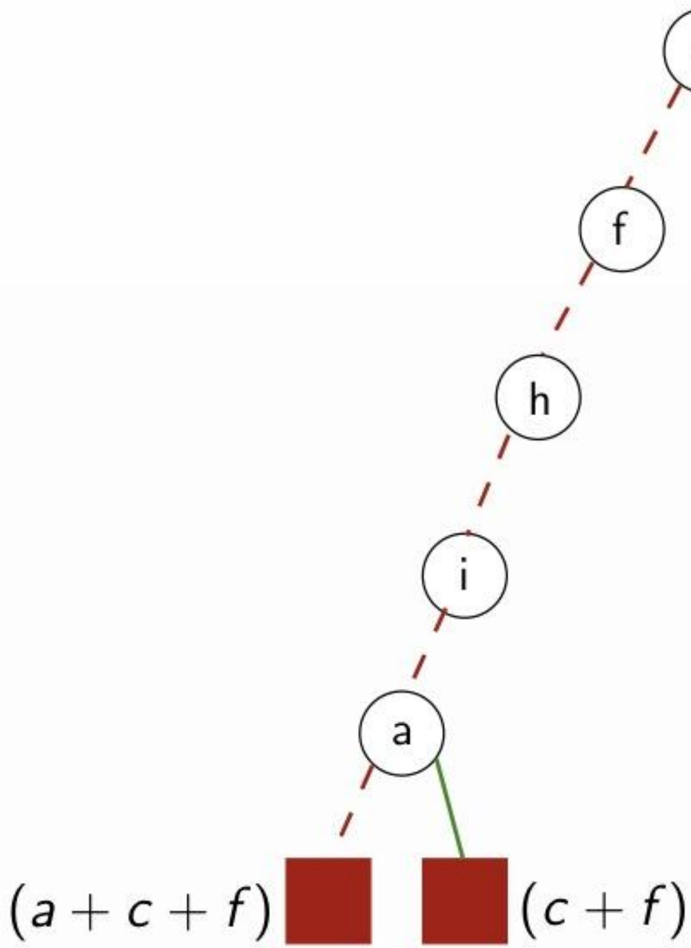
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

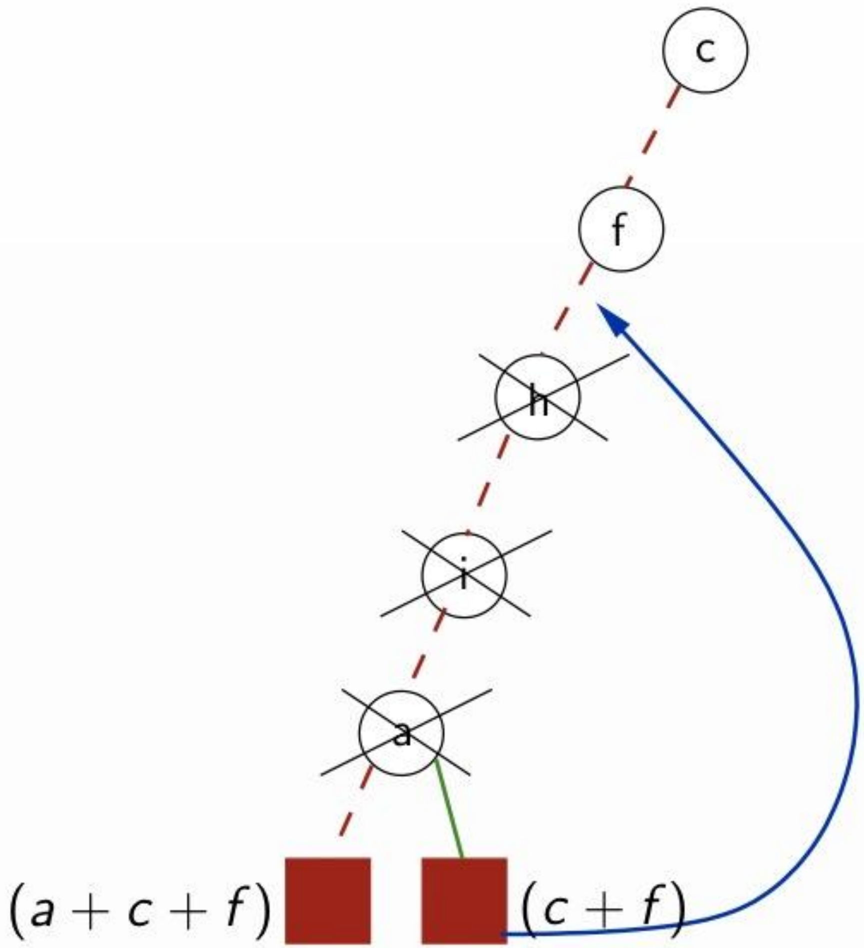
$$\begin{aligned}\varphi \;=\;& (a \vee b) \wedge (\neg b \vee \boxed{c} \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee \boxed{f}) \wedge \\ & (a \vee \boxed{c} \vee \boxed{f}) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (c = 1) \vee (f = 1)$
  - Learn new clause $(c \vee f)$

Non-Chronological Backtracking



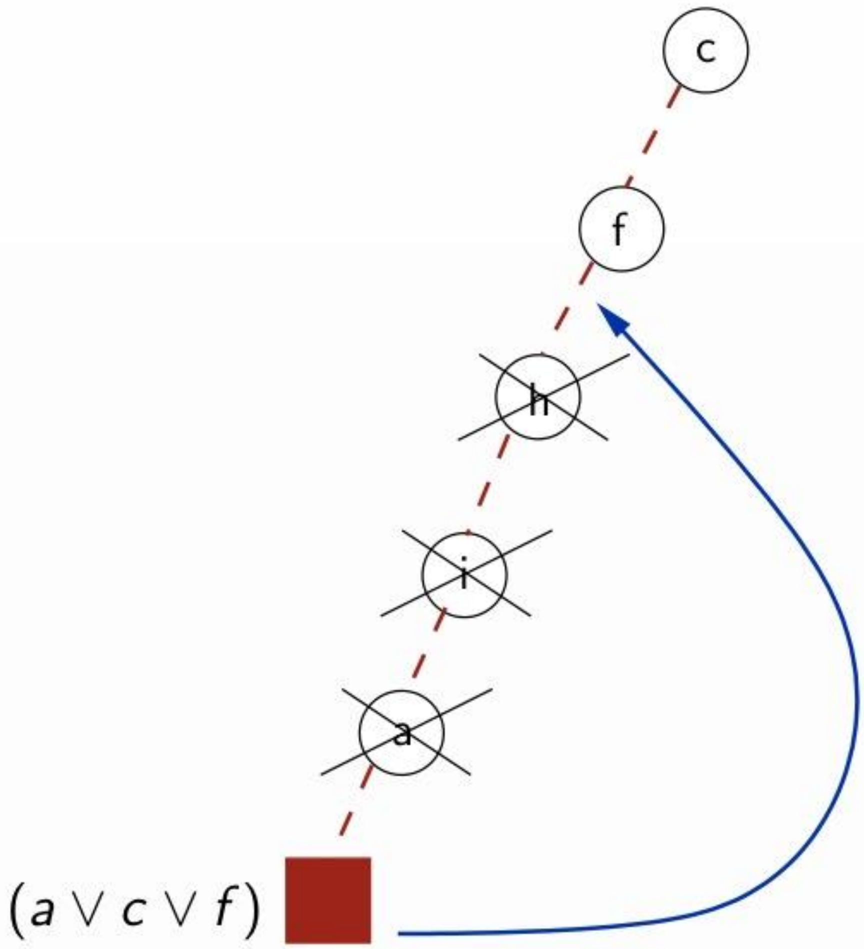
Non-Chronological Backtracking



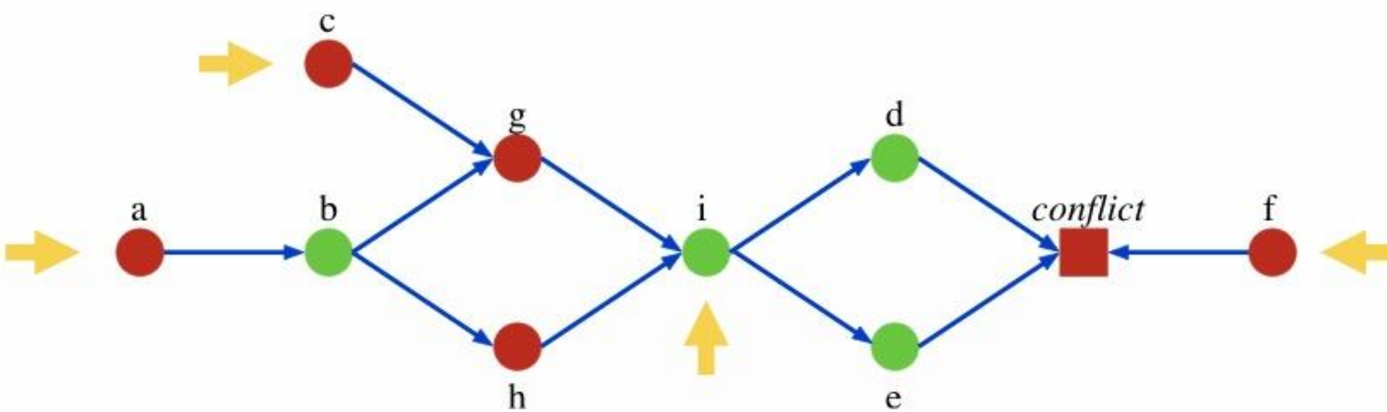
- Learnt clause: $(c \vee f)$
- Need to backtrack, given new clause
- Backtrack to most recent decision: $f = 0$

- Clause learning and non-chronological backtracking are hallmarks of modern SAT solvers

Most Recent Backtracking Scheme



Unique Implication Points (UIPs)



- Exploit structure from the implication graph
  - To have a more aggressive backtracking policy
- Identify additional clauses to learn [Marques-Silva&Sakallah'96]
  - Create clauses $(a \vee c \vee f)$ and $(\neg i \vee f)$
  - Imply not only $a = 1$ but also $i = 0$
- 1st UIP scheme is the most effective [Zhang et al.'01]
  - Create only one clause $(\neg i \vee f)$
  - Avoid creating similar clauses involving the same literals

Clause deletion policies

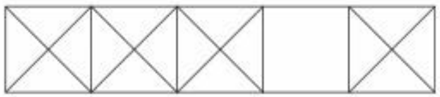
- Keep only the small clauses [Marques-Silva&Sakallah'96]
  - For each conflict record one clause
  - Keep clauses of size $\leq K$
  - Large clauses get deleted when become unresolved
- Keep only the relevant clauses [Bayardo&Schrag'97]
  - Delete unresolved clauses with $\leq M$ free literals
- Keep only the clauses that are used [Goldberg&Novikov'02]
  - Keep track of clauses activity

Data Structures

- Key point: only unit and unsatisfied clauses *must* be detected during search
  - Formula is unsatisfied when at least one clause is unsatisfied
  - Formula is satisfied when all the variables are assigned and there are no unsatisfied clauses
- In practice: unit and unsatisfied clauses may be identified using only two references
- Standard data structures (adjacency lists):
  - Each variable x keeps a reference to all clauses containing a literal in x
- Lazy data structures (watched literals):
  - For each clause, only two variables keep a reference to the clause, i.e. only 2 literals are watched

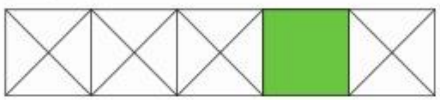
Standard Data Structures (adjacency lists)

literals0 = 4
literals1= 0
size = 5



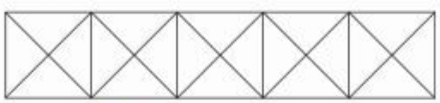
unit

literals0 = 4
literals1= 1
size = 5



satisfied

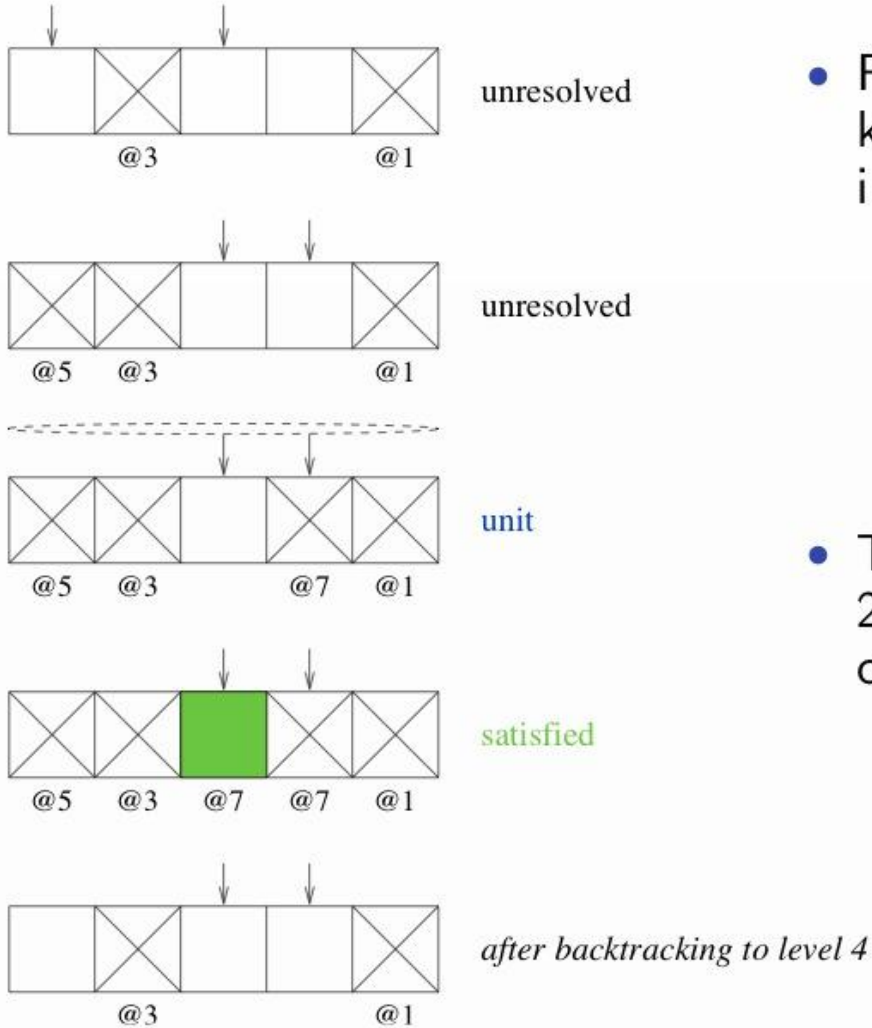
literals0 = 5
literals1= 0
size = 5



unsatisfied

- Each variable x keeps a reference to all clauses containing a literal in x
  - If variable x is assigned, then all clauses containing a literal in x are evaluated
  - If search backtracks, then all clauses of all newly unassigned variables are updated
- Total number of references is L, where L is the number of literals

Lazy Data Structures (watched literals)



- For each clause, only two variables keep a reference to the clause, i.e. only 2 literals are watched
  - If variable x is assigned, only the clauses where literals in x are watched need to be evaluated
  - If search backtracks, then nothing needs to be done
- Total number of references is $2 \times C$, where C is the number of clauses
  - In general $L \gg 2 \times C$, in particular if clauses are learnt

BCP Algorithm (1/8)

- What "causes" an implication? When can it occur?
  - All literals in a clause but one are assigned to False
    - (v1 + v2 + v3): implied cases: (0 + 0 + v3) or (0 + v2 + 0) or (v1 + 0 + 0)
  - For an N-literal clause, this can only occur after N-1 of the literals have been assigned to False
  - So, (theoretically) we could completely ignore the first N-2 assignments to this clause
  - In reality, we pick two literals in each clause to "watch" and thus can ignore any assignments to the other literals in the clause.
    - Example: (v1 + v2 + v3 + v4 + v5)
    - (**v1=X + v2=X** + v3=? {i.e. X or 0 or 1} + v4=? + v5=?)

BCP Algorithm (1.1/8)

- Big Invariants
  - Each clause has two watched literals.
  - If a clause can become unit via any sequence of assignments, then this sequence will include an assignment of one of the watched literals to F.
    - Example again: (v1 + v2 + v3 + v4 + v5)
    - (**v1=X** + **v2=X** + v3=? + v4=? + v5=?)
- BCP consists of identifying unit (and conflict) clauses (and the associated implications) while maintaining the "Big Invariants"

BCP Algorithm (2/8)

- Let's illustrate this with an example:

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3′
v1 + v2′
v1′+ v4
v1′
```

BCP Algorithm (2.1/8)

- Let's illustrate this with an example:

watched literals

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1'+ v4
v1'
```

One literal clause breaks invariants: handled as a special case (ignored hereafter)

- Initially, we identify any two literals in each clause as the watched ones
- Clauses of size one are a special case

BCP Algorithm (3/8)

- We begin by processing the assignment v1 = F (which is implied by the size one clause)

```
State:(v1=F)

Pending:
```

```
v2 + v3 + v1 + v4 + v5

v1 + v2 + v3'

v1 + v2'

v1'+ v4
```

BCP Algorithm (3.1/8)

- We begin by processing the assignment v1 = F (which is implied by the size one clause)

```
State:(v1=F)

Pending:
```

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1' + v4
```

- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to F.

BCP Algorithm (3.2/8)

- We begin by processing the assignment v1 = F (which is implied by the size one clause)

```
State:(v1=F)

Pending:
```

```
   v2 + v3 + v1 + v4 + v5
   v1 + v2 + v3'
   v1 + v2'
⇨  v1'+ v4
```

- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to F.
- We need not process clauses where a watched literal has been set to T, because the clause is now satisfied and so can not become unit.

BCP Algorithm (3.3/8)

- We begin by processing the assignment v1 = F (which is implied by the size one clause)

```
State:(v1=F)

Pending:
```

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1'+ v4
```

- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to F.
- We need not process clauses where a watched literal has been set to T, because the clause is now satisfied and so can not become unit.
- We *certainly* need not process any clauses where neither watched literal changes state (in this example, where v1 is not watched).

BCP Algorithm (4/8)

- Now let's actually process the second and third clauses:

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1'+ v4
```

```
State: (v1=F)
Pending:
```

BCP Algorithm (4.1/8)

- Now let's actually process the second and third clauses:

v2 + v3 + v1 + v4 + v5	v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'	v1 + v2 + v3'
v1 + v2'	v1 + v2'
v1' + v4	v1' + v4

```
State: (v1=F)
Pending:
```

```
State: (v1=F)
Pending:
```

- For the second clause, we replace v1 with v3' as a new watched literal. Since v3' is not assigned to F, this maintains our invariants.

BCP Algorithm (4.2/8)

- Now let's actually process the second and third clauses:

v2 + v3 + v1 + v4 + v5	v2 + v3 + v1 + v4 + v5
v1 + v2 + v3′	v1 + v2 + v3′
v1 + v2′	v1 + v2′
v1′ + v4	v1′ + v4
State: (v1=F) Pending:	State: (v1=F) Pending: (v2=F)

- For the second clause, we replace v1 with v3' as a new watched literal. Since v3' is not assigned to F, this maintains our invariants.
- The third clause is unit. We record the new implication of v2', and add it to the queue of assignments to process. Since the clause cannot again become unit, our invariants are maintained.

BCP Algorithm (5/8)

- Next, we process v2’. We only examine the first 2 clauses.

v2 + v3 + v1 + v4 + v5	v2 + v3 + v1 + v4 + v5
v1 + v2 + v3’	v1 + v2 + v3’
v1 + v2’	v1 + v2’
v1’ + v4	v1’ + v4
State: (v1=F, v2=F) Pending:	State: (v1=F, v2=F) Pending: (v3=F)

- For the first clause, we replace v2 with v4 as a new watched literal. Since v4 is not assigned to F, this maintains our invariants.
- The second clause is unit. We record the new implication of v3’, and add it to the queue of assignments to process. Since the clause cannot again become unit, our invariants are maintained.

BCP Algorithm (6/8)

- Next, we process v3'. We only examine the first clause.

v2 + v3 + v1 + v4 + v5	v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'	v1 + v2 + v3'
v1 + v2'	v1 + v2'
v1' + v4	v1' + v4
State: (v1=F, v2=F, v3=F) Pending:	State: (v1=F, v2=F, v3=F) Pending:

- For the first clause, we replace v3 with v5 as a new watched literal. Since v5 is not assigned to F, this maintains our invariants.
- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. Both v4 and v5 are unassigned. Let's say we decide to assign v4=T and proceed.

BCP Algorithm (7/8)

- Next, we process v4. We do nothing at all.

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1'+ v4
```

```
State:(v1=F, v2=F, v3=F,
v4=T)
```

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1'+ v4
```

```
State:(v1=F, v2=F, v3=F,
v4=T)
```

- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. Only v5 is unassigned. Let's say we decide to assign v5=F and proceed.

BCP Algorithm (8/8)

- Next, we process v5=F. We examine the first clause.

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1'+ v4
```

```
State:(v1=F, v2=F, v3=F,
v4=T, v5=F)
```

```
v2 + v3 + v1 + v4 + v5
v1 + v2 + v3'
v1 + v2'
v1'+ v4
```

```
State:(v1=F, v2=F, v3=F,
v4=T, v5=F)
```

- The first clause is already satisfied by v4 so we ignore it.
- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. No variables are unassigned, so the instance is SAT, and we are done.

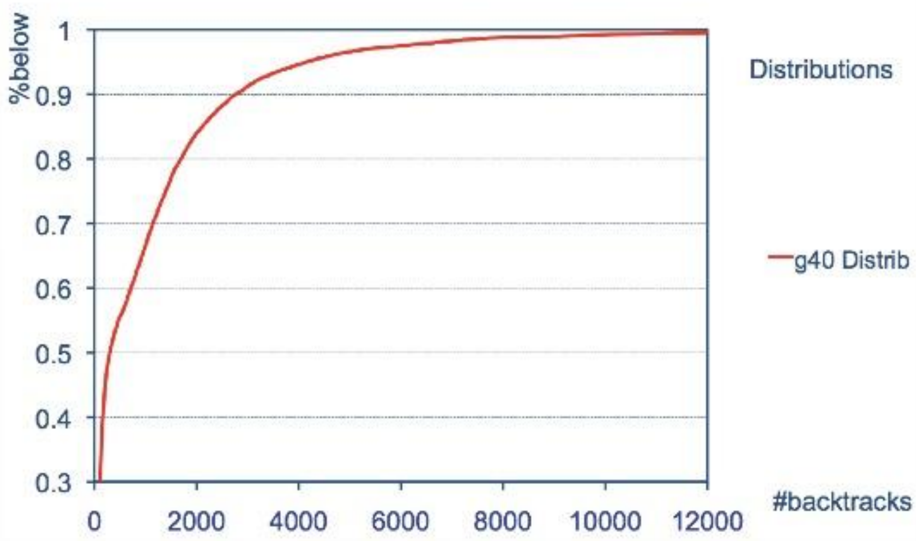
BCP Algorithm Summary

- During forward progress: Decisions and Implications
  - Only need to examine clauses where watched literal is set to F
    - Can ignore any assignments of literals to T
    - Can ignore any assignments to non-watched literals
- During backtrack: Unwind Assignment Stack
  - Any sequence of chronological unassignments will maintain our invariants
    - *So no action is required at all to unassign variables.*
- Overall
  - Minimize clause access

Search Heuristics

- Standard data structures: heavy heuristics
  - DLIS: Dynamic Large Individual Sum [Marques-Silva'99]
    - Selects the literal that appears most frequently in unresolved clauses
- Lazy data structures: light heuristics
  - VSIDS: Variable State Independent Decaying Sum [Moskewicz et al.'01]
    - Each literal has a counter, initialized to zero
    - When a new clause is recorded, the counter associated with each literal in the clause is incremented
    - The unassigned literal with the highest counter is chosen at each decision
  - Examples of variants
    - Counters updated also for literals in the clauses involved in conflicts [Goldberg&Novikov'02]

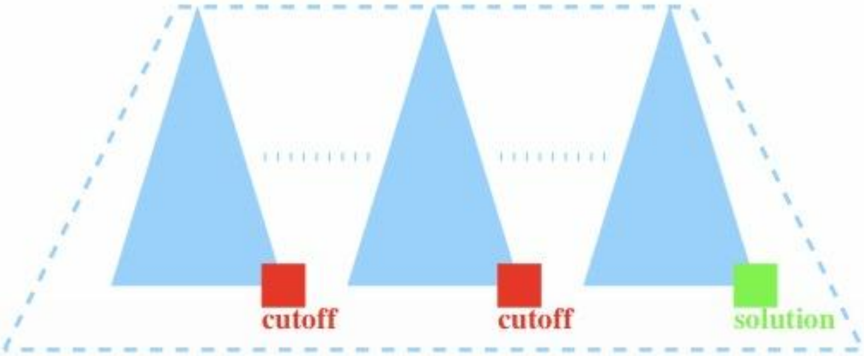
Restarts I



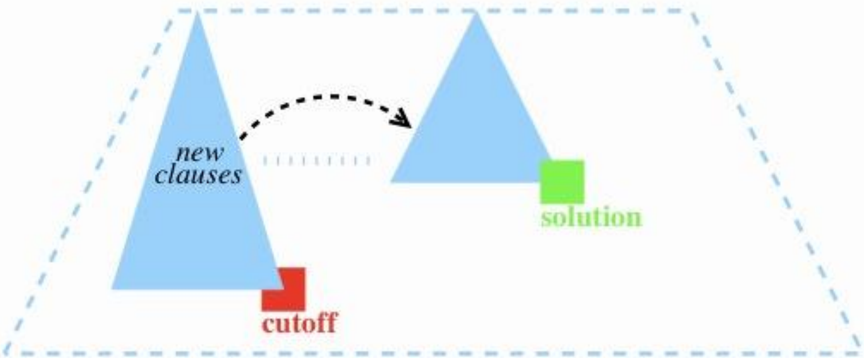
- Plot for processor verification instance with branching randomization and 10000 runs
  - More than 50% of the runs require less than 1000 backtracks
  - A small percentage requires more than 10000 backtracks
- Run times of backtrack search SAT solvers characterized by heavy-tail distributions

[Gomes et al.'98]

Restarts II

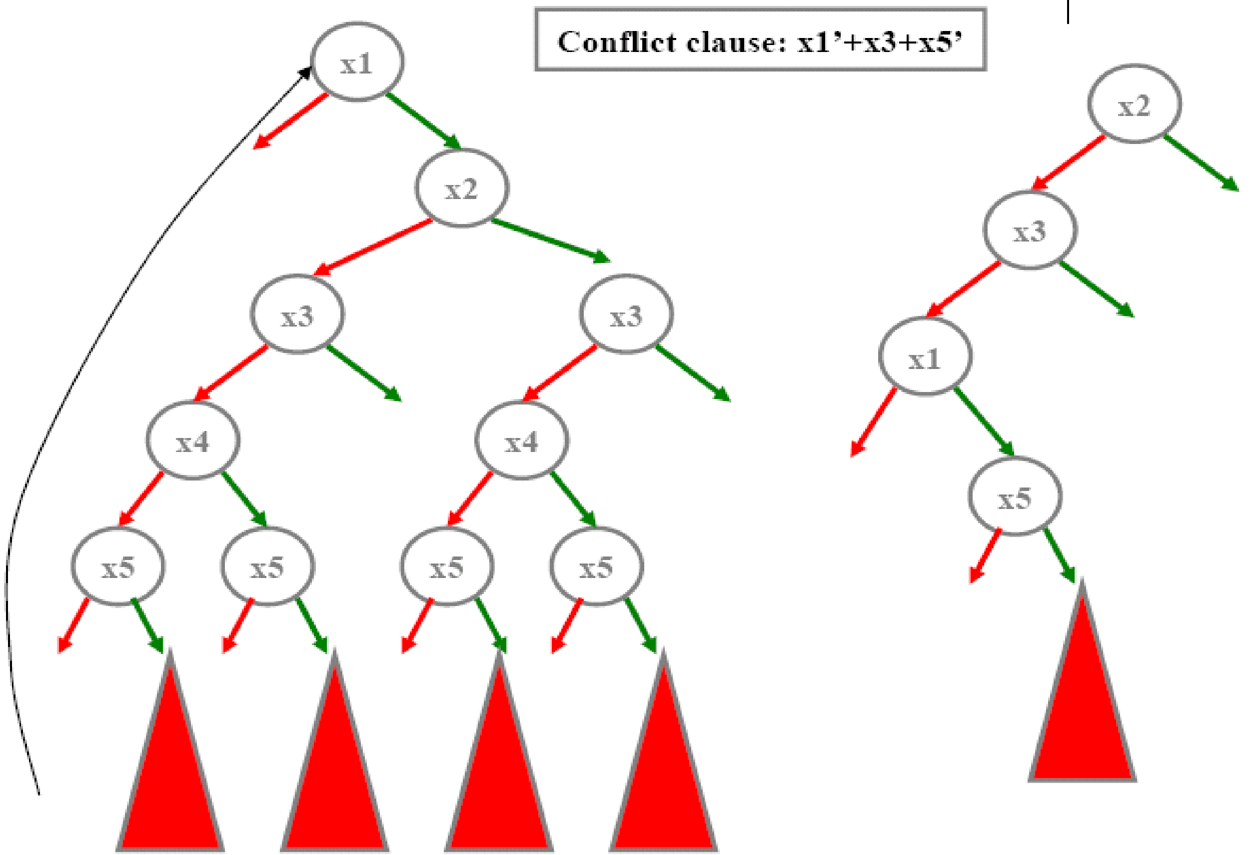


- Repeatedly restart the search each time a cutoff is reached
  - Randomization allows to explore different paths in search tree
- Resulting algorithm is incomplete
  - Increase the cutoff value
  - Keep clauses from previous runs



Restart

- Abandon the current search tree and reconstruct a new one
- Helps reduce variance - adds to robustness in the solver
- The clauses learned prior to the restart are *still there* after the restart and can help pruning the search space



Evolution of SAT Solvers

Instance	Posit'94	Grasp'96	Chaff'03	Minisat'03	Picosat'08
ssa2670-136	13.57	0.22	0.02	0.00	0.01
bf1355-638	310.93	0.02	0.02	0.00	0.03
design_3	> 1800	3.93	0.18	0.17	0.93
design_1	> 1800	34.55	0.35	0.11	0.68
4pipe_4_ooo	> 1800	> 1800	17.47	110.97	44.95
fifo8_300	> 1800	> 1800	348.50	53.66	39.31
w08_15	> 1800	> 1800	> 1800	99.10	71.89
9pipe_9_ooo	> 1800	> 1800	> 1800	> 1800	> 1800
c6288	> 1800	> 1800	> 1800	> 1800	> 1800

- Modern SAT algorithms can solve instances with hundreds of thousands of variables and tens of millions of clauses

Benchmarks

- Random
- Crafted
- Industrial

Random SAT specialty, the winners

1. ranov
2. g2wsat
3. vw

Random SAT specialty, the winners

Solver	Score	SAT answers	UNSAT answers
ranov	163903	209	0
g2wsat	101286	178	0
vw	76002	170	0
adaptnovelty	21748	119	0
saps	15603	104	0
kcnfs-2004	14604	92	0
dSatz-1a	8943	68	0
march-dl	7444	56	0
wllsatv1	7202	59	0
satELiteGTI	5198	46	0
minisat	5147	45	0

Random SAT+UNSAT specialty, the complete ranking

Solver	Score	SAT answers	UNSAT answers
kcnfs-2004	95075	92	75
march-dl	27141	56	43
dSatz-1a	22940	68	50
wllsatv1	16145	59	45
satELiteGTI	10074	46	33
minisat	10058	45	33

Crafted SAT+UNSAT specialty, the complete ranking

Solver	Score	SAT answers	UNSAT answers
vallst	56445	138	100
satELiteGTI	53128	122	126
march-dl	52432	138	99
minisat	43691	122	121
hsat-1	39497	130	90
csat	38324	113	112
zchaff	27455	112	89
zchaff-rand	24171	107	78
tts-3-0	21298	5	54
jerusat-A	19632	104	77

SAT+UNSAT specialty, the complete ranking

Solver	Score	SAT answers	UNSAT answers
ssatELiteGTI	99662	180	87
minisat	69485	166	84
haifaSat	50931	151	91
zchaff-rand	50515	132	94
jerusat-B	47487	163	80
csat	36526	140	91
compsat	25399	114	75
zchaff	31702	121	76
sat4j	21097	110	70
hsat-5	20995	99	54
vallst	16874	85	69
wllsatv1	12467	86	6

Qualified Solvers

Solver	Author	Affiliation
Actin (minisat+i)	Raihan Kibria	TU Darmstadt
Barcelogic	Robert Nieuwenhuis	TU Catalonia, Barcelona
Cadence MiniSAT	Niklas Een	Cadence Design Systems
CompSAT	Armin Biere	JKU Linz
Eureka	Alexander Nadel	Intel
HyperSAT	Domagoj Babic	UBC
MiniSAT 2.0	Niklas Sörensson	Chalmers
Mucsat	Nicolas Rachinsky	LMU Munich
MXC v.1	David Mitchell	SFU
PicoSAT	Armin Biere	JKU Linz
QCompSAT	Armin Biere	JKU Linz
QPicoSAT	Armin Biere	JKU Linz
Rsat	Thammanit Pipatsrisawat	UCLA
SAT4J	Daniel Le Berre	CRIL-CNRS
TINISAT	Jinbo Huang	NICTA
zChaff 2006	Zhaohui Fu	Princeton

SAT race 2006

Complete Ranking

Rank	Solver	Author	Affiliation	#solved	Speed Points	Total Score
1	MiniSAT 2.0	Niklas Sörensson	Chalmers	73	9.71	**82.71**
2	Eureka	Alexander Nadel	Intel	67	13.87	**80.87**
3	Rsat	Thammanit Pipatsrisawat	UCLA	72	8.45	**80.45**
4	Cadence MiniSAT	Niklas Een	Cadence Design Systems	63	6.39	**69.39**
5	Actin (minisat+i)	Raihan Kibria	TU Darmstadt	63	6.29	**69.29**
6	Barcelogic	Robert Nieuwenhuis	TU Catalonia, Barcelona	59	5.98	**64.98**
7	PicoSAT	Armin Biere	JKU Linz	57	5.00	**62.00**
8	QPicoSAT	Armin Biere	JKU Linz	54	5.39	**59.39**
9	TINISAT	Jinbo Huang	NICTA	54	4.91	**58.91**
10	SAT4J	Daniel Le Berre	CRIL-CNRS	49	4.20	**53.20**
11	QCompSAT	Armin Biere	JKU Linz	39	3.22	**42.22**
12	zChaff 2006	Zhaohui Fu	Princeton	38	3.78	**41.78**
13	CompSAT	Armin Biere	JKU Linz	38	3.21	**41.21**
14	MXC v.1	David Mitchell	SFU	29	2.23	**31.23**
15	Mucsat	Nicolas Rachinsky	LMU Munich	28	2.09	**30.09**
16	HyperSAT	Domagoj Babic	UBC	27	2.99	**29.99**

SAT race 2006

Outline

Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:

Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:

Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)

Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)

Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)
  - Done if all clauses satisfied

Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)
  - Done if all clauses satisfied
- Repeat (random) selection of assignment a number of times