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

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
  \((x_1 \lor \neg x_2 \lor \neg x_3)\)

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

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

- 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:
    \[ \varphi = (\neg x_1 \lor x_2) \land (x_3 \lor \neg x_2) \land (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]
    - \(x_1\) 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:
    \[ \varphi = (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

- A reference technique until the mid 90s; nowadays seldom used
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]

  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
  \]

[Davis & Putnam, JACM '60]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]

  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
  \]
**Unit Propagation**

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
  \]

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

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

\[
\text{RES}(x \lor \alpha, \neg x \lor \beta) = (\alpha \lor \beta)
\]

- Resolution forms the basis of a complete algorithm for SAT
  - Iteratively apply the following steps:
    - Select variable \(x\)
    - Apply resolution rule between every pair of clauses of the form \((x \lor \alpha)\) and \((\neg x \lor \beta)\)
    - 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

[Davis&Putnam, JACM'60]
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]
Resolution – An Example

$$(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash$$

$$\neg (x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash$$
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]
Resolution – An Example

\[ (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash \]

\[ (\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \]

\[ (x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \]

\[ (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \]
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(x_3)\]

- 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
    - Resolution
    - Stalmarck’s method
    - Recursive learning
    - ...
  - Binary Decision Diagrams (BDDs)
  - Backtrack search / DPLL
    - Conflict-Driven Clause Learning (CDCL)
  - ...

[e.g. Huth & Ryan’04]
Outline

Preliminaries

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

SAT-Based Modelling
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

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \land \\
\text{conflict} \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \land \\
\text{conflict} \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

$$\varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land$$
$$\land (\neg b \lor \neg d \lor \neg e) \land$$
$$\land (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land$$
$$\land (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e)$$

conflict
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land (\neg b \lor \neg d \lor \neg e) \land (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]

Comparing with CSP:
- Sat can be decided before all variables are assigned

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

Think Horn clauses
Outline

Preliminaries

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

SAT-Based Modelling
CDCL SAT Solvers – Basic Techniques

- Based on DPLL
  - Must be able to prove unsatisfiability
- New clauses are learned from conflicts
  - Backtracking can be non-chronological
- Structure of conflicts is exploited (UIPs)
- Backtrack search is periodically restarted
- Lazy data structures are used
  - Compact with low maintenance overhead
- Branching is guided by conflicts
  - E.g. VSIDS, etc.

[Davis et al., JACM’60, CACM’62]
[Marques-Silva&Sakallah, ICCAD’96]
[Marques-Silva&Sakallah, ICCAD’96]
[Gomes et al., AAAI’98]
[Moskewicz et al, DAC’01]
[Moskewicz et al, DAC’01]
CDCL SAT Solvers – Additional Techniques

- (Currently) **effective** techniques:
  - Unused learned clauses are discarded
  - Use formula preprocessing I
  - Minimize learned clauses
  - Use literal progress saving
  - Use dynamic restart policies
  - Exploit **extended implication graphs**
  - Identify **glue clauses**

- (Currently) **ineffective** techniques:
  - Identify pure literals
  - Implement variable lookahead
  - Use formula preprocessing II

[Goldberg & Novikov, DATE ‘02]
[Een & Biere, SAT ‘05]
[Sorensson & Biere, SAT ‘09]
[Pipatsrisawat & Darwiche, SAT ‘07]
[Biere, SAT ‘08]
[Audemard et al., SAT ‘08]
[Audemard & Simon, IJCAI ‘09]

[Davis & Putnam, JACM ‘60]
[Anbulagan & Li, IJCAI ‘97]
[Brafman, IJCAI ‘01]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied

  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

  \[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

  \[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4) \\
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
  \]

- **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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- 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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- 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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- 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

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots$$

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

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \((\neg d \lor \neg e \lor f)\) is unsatisfied
- \((a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0)\)
Clause Learning

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

\[
\varphi = \left( a \lor b \right) \land \left( \neg b \lor c \lor d \right) \land \left( \neg b \lor e \right) \land \left( \neg d \lor \neg e \lor f \right) \ldots
\]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \( \neg d \lor \neg e \lor f \) is unsatisfied
- \((a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0)\)
- \((\varphi = 1) \Rightarrow (a = 1) \lor (c = 1) \lor (f = 1)\)
During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict:

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f)$$

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

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land \\
(a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- 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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- 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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- 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

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)$$

- 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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land \\
(a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- 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
Non-Chronological Backtracking

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

$$
\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land \\
(a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)
$$

- 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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- 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
- \((c = 0) \land (f = 0) \Rightarrow (\varphi = 0)\)
- \((\varphi = 1) \Rightarrow (c = 1) \lor (f = 1)\)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- 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
- \( (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (c = 1) \lor (f = 1) \)

- Learn new clause \( (c \lor f) \)
Non-Chronological Backtracking

\[(a + c + f) \quad (c + f)\]
Non-Chronological Backtracking

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

\((a + c + f)\)  \(\rightarrow\)  \((c + f)\)

Clause learning and non-chronological backtracking are hallmarks of modern SAT solvers.
Most Recent Backtracking Scheme

\[(a \lor c \lor f)\]
Unique Implication Points (UIPs)

- Exploit **structure** from the implication graph
  - To have a more aggressive backtracking policy
- Identify **additional clauses** to learn
  - Create clauses \((a \lor c \lor f)\) and \((\neg i \lor f)\)
  - Imply not only \(a = 1\) but also \(i = 0\)
- 1st UIP scheme is the most effective
  - Create only one clause \((\neg i \lor f)\)
  - Avoid creating similar clauses involving the same literals

[Marques-Silva & Sakallah ’96]

[Zhang et al. ’01]
Clause deletion policies

- Keep only the **small clauses**  
  - For each conflict record one clause  
  - Keep clauses of size $\leq K$  
  - Large clauses get deleted when become unresolved

- Keep only the **relevant clauses**  
  - Delete unresolved clauses with $\leq M$ free literals

- Keep only the clauses **that are used**  
  - 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)

- 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

\[ \text{unresolved} \]
\[ \text{unresolved} \]
\[ \text{unit} \]
\[ \text{satisfied} \]
\[ \text{after backtracking to level 4} \]
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:

\[ v_2 + v_3 + v_1 + v_4 + v_5 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1 + v_2' \]
\[ v_1' + v_4 \]
\[ v_1' \]
Let’s illustrate this with an example:

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 $v_1 = F$ (which is implied by the size one clause)

<table>
<thead>
<tr>
<th>State: $(v_1=F)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending:</td>
</tr>
</tbody>
</table>

\[ v_2 + v_3 + v_1 + v_4 + v_5 \]

\[ v_1 + v_2 + v_3' \]

\[ v_1 + v_2' \]

\[ v_1' + v_4 \]
BCP Algorithm (3.1/8)

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

\[
\begin{align*}
\text{State: } & (v1=F) \\
\text{Pending:} & \\
\implies & (v1 + v2 + v3') \\
\implies & (v1 + v2') \\
\implies & (v1' + v4)
\end{align*}
\]

- 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)

\[
\begin{align*}
\text{State: } (v1=F) \\
\text{Pending:} \\
\end{align*}
\]

\[
\begin{align*}
&v2 + v3 + v1 + v4 + v5 \\
&v1 + v2 + v3' \\
&v1 + v2' \\
&v1' + v4 \\
\end{align*}
\]

- 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)

\[
\rightarrow \quad v2 + v3 + v1 + v4 + v5
\]

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

- 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:

\[ v_2 + v_3 + v_1 + v_4 + v_5 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1 + v_2' \]
\[ v_1' + v_4 \]

State: \((v_1=F)\)
Pending:
BCP Algorithm (4.1/8)

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

\[
\begin{align*}
&v_2 + v_3 + v_1 + v_4 + v_5 \\
v_1 + v_2 + v_3' \\
v_1 + v_2' \\
v_1' + v_4 \\
\end{align*}
\]

\[
\begin{align*}
&v_2 + v_3 + v_1 + v_4 + v_5 \\
v_1 + v_2 + v_3' \\
v_1 + v_2' \\
v_1' + v_4 \\
\end{align*}
\]

State: (v_1=F)  
Pending:

State: (v_1=F)  
Pending:

- For the second clause, we replace v_1 with v_3’ as a new watched literal. Since v_3’ is not assigned to F, this maintains our invariants.
BCP Algorithm (4.2/8)

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

\[
\begin{align*}
v2 + v3 + v1 + v4 + v5 \\
v1 + v2 + v3' \\
v1 + v2' \\
v1' + v4
\end{align*}
\]

\[
\begin{align*}
v2 + v3 + v1 + v4 + v5 \\
v1 + v2 + v3' \\
v1 + v2' \\
v1' + v4
\end{align*}
\]

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

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

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

\[
\begin{align*}
v2 &+ v3 &+ v1 &+ \textcolor{red}{v4} &+ v5 \\
v1 &+ \textcolor{red}{v2} &+ v3' \\
v1 &+ \textcolor{red}{v2'} \\
v1' &+ v4
\end{align*}
\]

\[
\begin{align*}
v2 &+ v3 &+ v1 &+ \textcolor{red}{v4} &+ v5 \\
v1 &+ \textcolor{red}{v2} &+ v3' \\
v1 &+ \textcolor{red}{v2'} \\
v1' &+ v4
\end{align*}
\]

\[
\text{State: (v1=F, v2=F, v3=F, v4=T)}
\]

\[
\text{State: (v1=F, v2=F, v3=F, v4=T)}
\]

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.
Next, we process $v_5 = F$. We examine the first clause.

$\begin{align*}
\text{State:} (v_1 = F, v_2 = F, v_3 = F, v_4 = T, v_5 = F)
\end{align*}$

The first clause is already satisfied by $v_4$ 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]
• 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]
Repeats 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

Conflicts clause: $x_1' + x_3 + x_5'$
## Evolution of SAT Solvers

<table>
<thead>
<tr>
<th>Instance</th>
<th>Posit’94</th>
<th>Grasp’96</th>
<th>Chaff’03</th>
<th>Minisat’03</th>
<th>Picosat’08</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssa2670-136</td>
<td>13.57</td>
<td>0.22</td>
<td>0.02</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>bf1355-638</td>
<td>310.93</td>
<td>0.02</td>
<td>0.02</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td>design_3</td>
<td>&gt; 1800</td>
<td>3.93</td>
<td>0.18</td>
<td>0.17</td>
<td>0.93</td>
</tr>
<tr>
<td>design_1</td>
<td>&gt; 1800</td>
<td>34.55</td>
<td>0.35</td>
<td>0.11</td>
<td>0.68</td>
</tr>
<tr>
<td>4pipe_4_ooo</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>17.47</td>
<td>110.97</td>
<td>44.95</td>
</tr>
<tr>
<td>fifo8_300</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>348.50</td>
<td>53.66</td>
<td>39.31</td>
</tr>
<tr>
<td>w08_15</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>99.10</td>
<td>71.89</td>
</tr>
<tr>
<td>9pipe_9_ooo</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
</tr>
<tr>
<td>c6288</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>SAT answers</th>
<th>UNSAT answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ranov</td>
<td>163903</td>
<td>209</td>
<td>0</td>
</tr>
<tr>
<td>g2wsat</td>
<td>101286</td>
<td>178</td>
<td>0</td>
</tr>
<tr>
<td>vw</td>
<td>76002</td>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>adaptnovelty</td>
<td>21748</td>
<td>119</td>
<td>0</td>
</tr>
<tr>
<td>saps</td>
<td>15603</td>
<td>104</td>
<td>0</td>
</tr>
<tr>
<td>kcnfs-2004</td>
<td>14604</td>
<td>92</td>
<td>0</td>
</tr>
<tr>
<td>dSat-1a</td>
<td>8943</td>
<td>68</td>
<td>0</td>
</tr>
<tr>
<td>march-dl</td>
<td>7444</td>
<td>56</td>
<td>0</td>
</tr>
<tr>
<td>wllsatv1</td>
<td>7202</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>satELiteGTI</td>
<td>5198</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>minisat</td>
<td>5147</td>
<td>45</td>
<td>0</td>
</tr>
</tbody>
</table>
Random SAT+UNSAT specialty, the complete ranking

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>SAT answers</th>
<th>UNSAT answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>kcnfs-2004</td>
<td>95075</td>
<td>92</td>
<td>75</td>
</tr>
<tr>
<td>march-dl</td>
<td>27141</td>
<td>56</td>
<td>43</td>
</tr>
<tr>
<td>dSatz-1a</td>
<td>22940</td>
<td>68</td>
<td>50</td>
</tr>
<tr>
<td>wllsatv1</td>
<td>16145</td>
<td>59</td>
<td>45</td>
</tr>
<tr>
<td>satELiteGTI</td>
<td>10074</td>
<td>46</td>
<td>33</td>
</tr>
<tr>
<td>minisat</td>
<td>10058</td>
<td>45</td>
<td>33</td>
</tr>
</tbody>
</table>
Crafted SAT+UNSAT specialty, the complete ranking

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>SAT answers</th>
<th>UNSAT answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>vallst</td>
<td>56445</td>
<td>138</td>
<td>100</td>
</tr>
<tr>
<td>satELiteGI</td>
<td>53128</td>
<td>122</td>
<td>126</td>
</tr>
<tr>
<td>march-dl</td>
<td>52432</td>
<td>138</td>
<td>99</td>
</tr>
<tr>
<td>minisat</td>
<td>43691</td>
<td>122</td>
<td>121</td>
</tr>
<tr>
<td>hsat-1</td>
<td>39497</td>
<td>130</td>
<td>90</td>
</tr>
<tr>
<td>csat</td>
<td>38324</td>
<td>113</td>
<td>112</td>
</tr>
<tr>
<td>zchaff</td>
<td>27455</td>
<td>112</td>
<td>89</td>
</tr>
<tr>
<td>zchaff-rand</td>
<td>24171</td>
<td>107</td>
<td>78</td>
</tr>
<tr>
<td>tts-3-0</td>
<td>21298</td>
<td>5</td>
<td>54</td>
</tr>
<tr>
<td>jerusat-A</td>
<td>19632</td>
<td>104</td>
<td>77</td>
</tr>
</tbody>
</table>
# SAT+UNSAT specialty, the complete ranking

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

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

SAT race 2006
## Complete Ranking

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

Preliminaries

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

SAT-Based Modelling
Organization of Local Search

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

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]
Organization of Local Search

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

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- 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:
  
  \[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]
- 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:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- 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:
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]

- 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:
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]

- 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:
  
  \[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- 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