Advanced Programming in Artificial Intelligence

Josep Argelich
Degree in Computer Engineering

Outline

- Introduction
- Complete approaches
  - SAT Clause Tableaux
  - DPLL
- Improving DPLL
  - Variable selection heuristics
- Clause Learning
- Lazy Data Structures
- MaxSAT
  - Max-SAT branch and bound algorithms
  - Underestimations
  - Inference rules

Introduction

Algorithm Classification

No Systematic Algorithms (incomplete)

- Do not always find the solution
  - Even with a lot of time resources
- For decision problems
  - Solid, but not complete (cannot finish when there is no solution)
- For optimization problems
  - Solid, but not complete (gives you the sub-optimum)

Algorithm Classification

Systematic Algorithms (complete)

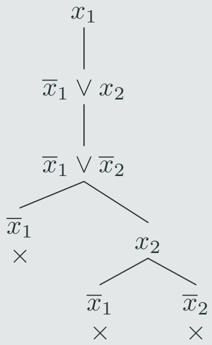
- Always find the solution
  - If they have enough resources (time)
- For decision problems
  - Solid (cannot prove wrong things, is correct) and complete (tells you if there is solution or not)
- For optimization problems
  - Solid and complete (gives you the optimum)

Complete approaches

SAT Clause Tableaux

Example

$\{x_1, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2\}$



Davis-Putnam (DP) algorithm

Given $F \equiv (A \vee p) \wedge (B \vee \neg p) \wedge R$

Davis Putnam Rule: $F \equiv (A \vee B) \wedge R$

The algorithm works as follows:

- for every variable in the formula
    - for every clause c containing the variable and every clause n containing the negation of the variable
        - *resolve* c and n and add the resolvent to the formula
    - remove all original clauses containing the variable or its negation

Davis-Putnam-Logemann-Loveland (DPLL) algorithm

- Davis, Putnam, Logemann, Loveland'62 proposed an improvement over DP
- Splitting the formula instead of applying resolution
- When a model is found, stop
- When a conflict in the search tree, backtrack
- It is a search algorithm for a model
- Best fitting algorithm: Depth First Search

Davis-Putnam-Logemann-Loveland (DPLL) algorithm

Output: Satisfiability of ϕ

Function `DavisLogemannLoveland(`ϕ : CNF formula`)` **: Boolean**

```
    UnitPropagation(ϕ)
    PureLiteralRule(ϕ)
    if ϕ = ∅ then return true

    if □ ∈ ϕ then return false

    ℓ ← literal in c ∈ ϕ
    return (DavisLogemannLoveland(ϕ ∧ ℓ) ∨
     DavisLogemannLoveland(ϕ ∧ ℓ̄))
end
```

Algorithm 1: `DavisLogemannLoveland`(ϕ)

Davis-Putnam-Logemann-Loveland (DPLL) algorithm

- Unit propagation is a key technique
- Pure literal detection can be eliminated
- At every level, the clause sizes are been reduced at least by one

Davis-Putnam-Logemann-Loveland (DPLL) algorithm

Example

The search tree for the CNF formula below is displayed in Figure.

$$(p_1 \vee p_5), (p_1 \vee \neg p_6), (p_1 \vee \neg p_2 \vee p_4), (p_1 \vee p_2 \vee \neg p_4), (\neg p_2 \vee \neg p_4),$$
$$(p_2 \vee p_4), (\neg p_1 \vee \neg p_2), (p_2 \vee p_3), (p_1 \vee p_2 \vee p_3)$$

Solid lines are for splitting assignments, and dashed lines for unit propagation and monotone literal assignments. Black nodes mark whenever a conflict is found.

Davis-Putnam-Logemann-Loveland (DPLL) algorithm

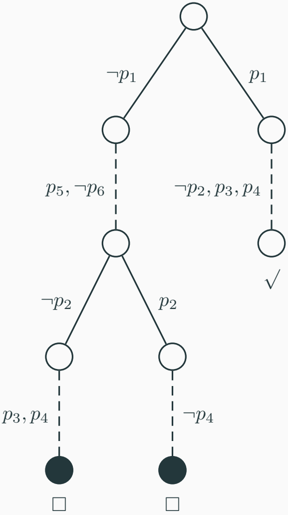
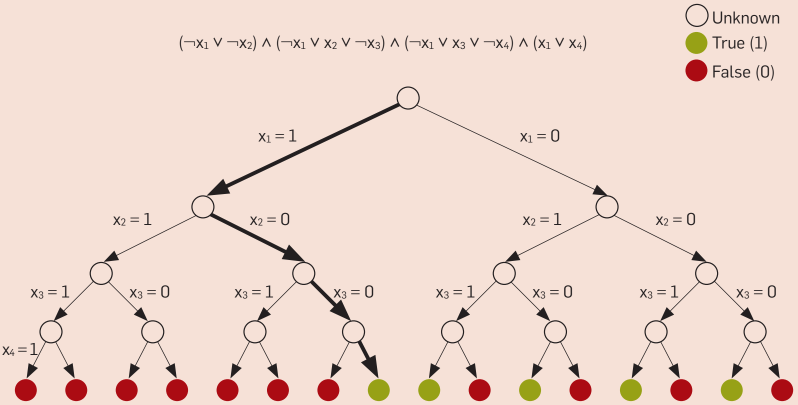


Figure 2.1: Search tree for DLL applied to Example 2.4.

Davis-Putnam-Logemann-Loveland (DPLL) algorithm

Figure 2. Search space of a formula.



Davis-Putnam-Logemann-Loveland (DPLL) algorithm

- It detects contradictions at internal nodes (empty clause).
- It detects models at internal nodes (empty formula).
- Empty clauses are found by unit propagation.
- Chronological backtracking.
- The algorithm is commonly known as *DPLL*.

Improving DPLL

Variable selection heuristics

Variable selection heuristics:

- MO: Variable with most occurrences
- MOMS: Most Often in Minimum Size (Pretolani '93)
- Select most equilibrated variable ($len(x) \times len(\neg x)$)
- Jeroslow-Wang $JW(x) = \sum 2^{-len(c)}$
- Satz:
  - Inspired in MOMS
  - $H(x) = w(x) \times w(\neg x) \times 2^{10} + w(x) + w(\neg x)$.
- VSIDS: later...

Variable selection heuristics

Variable with Most Occurrences (MO):

- We pick the variable that has most occurrences in the formula
- Easy to implement
- We can branch first on the sense that appears more (or not)
- It does not take into account the length of the clauses where the variable appears

Variable selection heuristics

Most Often in Minimum Size (MOMS):

- We pick the variable that occurs most often in clauses of minimum size
- As we do not have unit clauses (because UP), the best are the variables that appear in binary clauses
- Next, the variables that appear in ternary clauses and so on

Variable selection heuristics

Select most equilibrated variable:

- $\mathit{occurrences}(x) \times \mathit{occurrences}(\neg x)$
- First one that takes into account the polarization of the variable
- Most equilibrated variables have higher values
- Does not take into account the size of the clauses

Variable selection heuristics

Jeroslow-Wang (JW):

- $JW(x) = \sum 2^{-len(c)}$
- Occurrences of the variable in shorter clauses count more
- For instance, occurrences in binary clauses has double the value of occurrences in ternary clauses

Variable Selection Heuristic

Example

$$\Phi \equiv \begin{array}{l}(p_1 \vee p_5), (p_1 \vee \neg p_6), (p_1 \vee \neg p_2 \vee p_4), \\ (p_1 \vee p_2 \vee \neg p_4), (\neg p_2 \vee \neg p_4), (p_2 \vee p_4), \\ (p_2 \vee p_3), (p_1 \vee p_2 \vee p_3)\end{array}$$

Variable	Occurrences Binary	Occurrences Ternary
p_1		
p_2		
p_3		
p_4		

MOMS

$MOMS(\Phi) = p_2$

JW

$JW(\Phi) = p_2$

Example

$$\Phi \equiv \begin{array}{l} (p_1 \vee p_5), (p_1 \vee \neg p_6), (p_1 \vee \neg p_2 \vee p_4), \\ (p_1 \vee p_2 \vee \neg p_4), (\neg p_2 \vee \neg p_4), (p_2 \vee p_4), \\ (p_2 \vee p_3), (p_1 \vee p_2 \vee p_3) \end{array}$$

Variable	Occurrences Binary	Occurrences Ternary
p_1	2	3
p_2	3	3
p_3	1	1
p_4	2	2

MOMS

$MOMS(\Phi) = p_2$

JW

$JW(\Phi) = p_2$

Variable Selection Heuristic

Example

$$\Phi \equiv \begin{array}{l}(p_1 \vee p_5), (p_1 \vee \neg p_6), (p_1 \vee \neg p_2 \vee p_4),\\ (p_1 \vee p_2 \vee \neg p_4), (\neg p_2 \vee \neg p_4), (p_2 \vee p_4),\\ (p_2 \vee p_3), (p_1 \vee p_2 \vee p_3)\end{array}$$

Variable	Occurrences Binary	Occurrences Ternary
p_1	2	3
p_2	3	3
p_3	1	1
p_4	2	2

MOMS

$MOMS(\Phi) = p_2$

JW

$JW(\Phi) = p_2$

Variable Selection Heuristic

Example

$$\Phi \equiv \begin{array}{l} (p_1 \vee p_5), (p_1 \vee \neg p_6), (p_1 \vee \neg p_2 \vee p_4), \\ (p_1 \vee p_2 \vee \neg p_4), (\neg p_2 \vee \neg p_4), (p_2 \vee p_4), \\ (p_2 \vee p_3), (p_1 \vee p_2 \vee p_3) \end{array}$$

Variable	Occurrences Binary	Occurrences Ternary
p_1	2	3
p_2	3	3
p_3	1	1
p_4	2	2

MOMS

$MOMS(\Phi) = p_2$

JW

$JW(\Phi) = p_2$

Variable selection heuristics

Solver Satz method:

- Inspired in MOMS
- $H(x) = w(x) \times w(\neg x) \times 2^{10} + w(x) + w(\neg x)$
- $w(x) = \sum occurrence_x \times 2^{-len(c_x)}$
- Takes into account both, polarization of the variables and size of the clauses where the variables appear

Variable Selection Heuristic

Example

$$\Phi \equiv \begin{array}{l} (\neg p_1 \vee p_5), (p_1 \vee \neg p_6), (p_1 \vee \neg p_2 \vee p_4), \\ (p_1 \vee p_2 \vee \neg p_4), (\neg p_2 \vee \neg p_4), (p_2 \vee p_4), \\ (p_2 \vee p_3), (p_1 \vee p_2 \vee p_3) \end{array}$$

$$H(x) = w(x) \times w(\neg x) \times 2^{10} + w(x) + w(\neg x)$$
$$w(x) = \sum occurrence_x \times 2^{-len(c_x)}$$

$H(p_1) = ?$

Solver Satz

$H(p_1) = 0.625 \times 0.25 \times 2^{10} + 0.625 + 0.25 = 160 + 0.875 = 160.875$
$w(p_1) = 0.25 + 0.125 + 0.125 + 0.125 = 0.625$
$w(\neg p_1) = 0.25$

Variable Selection Heuristic

Example

$$\Phi \equiv \begin{array}{l}(\neg p_1 \vee p_5), (p_1 \vee \neg p_6), (p_1 \vee \neg p_2 \vee p_4),\\ (p_1 \vee p_2 \vee \neg p_4), (\neg p_2 \vee \neg p_4), (p_2 \vee p_4),\\ (p_2 \vee p_3), (p_1 \vee p_2 \vee p_3)\end{array}$$

$H(x) = w(x) \times w(\neg x) \times 2^{10} + w(x) + w(\neg x)$
$w(x) = \sum occurrence_x \times 2^{-len(c_x)}$

$H(p_1) = ?$

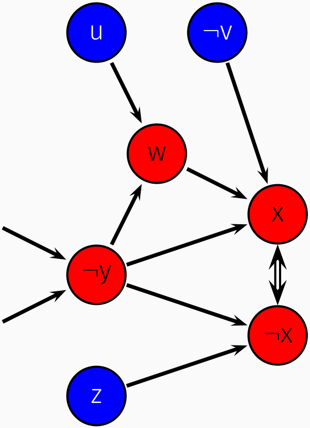
Solver Satz

$H(p_1) = 0.625 \times 0.25 \times 2^{10} + 0.625 + 0.25 = 160 + 0.875 = 160.875$
$w(p_1) = 0.25 + 0.125 + 0.125 + 0.125 = 0.625$
$w(\neg p_1) = 0.25$

Clause Learning

Clause learning

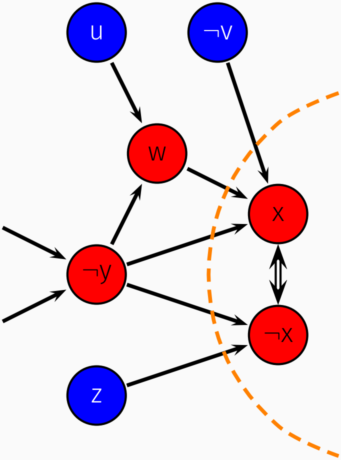
- Prunes parts of the search space without solutions
- Detects a conflict $\rightarrow$ analyzes it using a learning schema
- Adds new clauses to the formula
- There are several learning schema



Implication graph

decision variables
implied variables

$(\neg u \lor y \lor w)$
$(y \lor \neg z \lor \neg x)$
$(v \lor \neg w \lor y \lor x)$



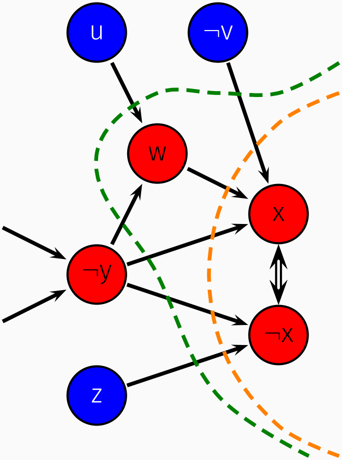
Implication graph

decision variables

implied variables

$(\neg u \vee y \vee w)$
$(y \vee \neg z \vee \neg x)$
$(v \vee \neg w \vee y \vee x)$

conflict $(v \vee \neg w \vee y \vee \neg z)$



Implication graph

decision variables
implied variables

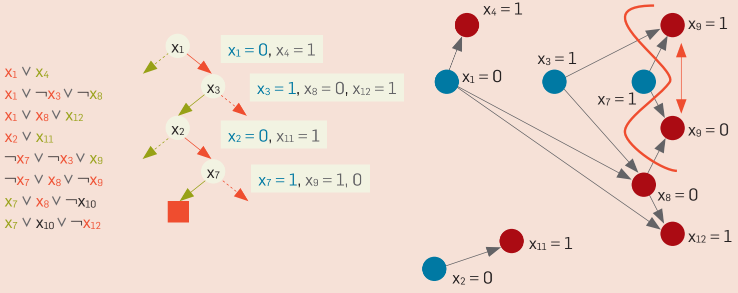
$(\neg u \lor y \lor w)$
$(y \lor \neg z \lor \neg x)$
$(v \lor \neg w \lor y \lor x)$

conflict $(v \lor \neg w \lor y \lor \neg z)$

1-UIP $(v \lor \neg u \lor \neg z \lor y)$

Learning schema

Figure 3. Conflict-driven learning and non-chronological backtracking.



$x_3 = 1 \wedge x_7 = 1 \wedge x_8 = 0 \longrightarrow$ conflict

$(\neg x_3 \vee \neg x_7 \vee \neg x_8)$ is valid

Clause learning

Now, as formula changes during the search...

- New clauses can stay forever
- After learning a clause, can we already skip parts of the search tree where we are?
- We can consider restarts?
- Rethink variable selection heuristics?

Clause learning

Now, as formula changes during the search...

- New clauses can stay forever
- After learning a clause, can we already skip parts of the search tree where we are?
- We can consider restarts?
- Rethink variable selection heuristics?

Restarts

- With new clauses the formula is different, so variable selection heuristics may choose other variables
- Variable selection heuristics have more information
- Branching order can be different and can lead to better performance

Clause learning

Now, as formula changes during the search...

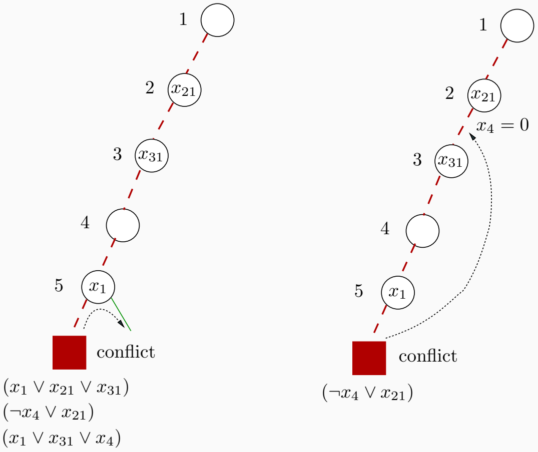
- New clauses can stay forever
- After learning a clause, can we already skip parts of the search tree where we are?
- We can consider restarts?
- Rethink variable selection heuristics?

Non-Chronological Backtracking

- After learning a clause, are we already on a conflicting part of the search space?
- If so, we can backtrack directly outside the conflict

Conflict Driven Clause Learning (CDCL)

Chapter 4. CDCL Solvers 137



(a) GRASP clause learning [MSS96] (b) Chaff clause learning [MMZ+01]

Figure 4.3. Alternative backtracking schemes

VSIDS Variable Selection Heuristic

Variable State Independent Decaying Sum (VSIDS)

- Collect statistics over learned clauses to guide the direction of the search (*activity*)
- Variables in recent learned clauses are favored
- Additive bumping
  - When a clause *c* is learned, *activity* of variables in *c* is increased (usually by 1)
- Multiplicative decay
  - At regular intervals, *activities* of all variables are multiplied by a constant α, where $0 < \alpha < 1$

Pros and Cons of Clause Learning

Pros

- Cheap compared to previous methods
- Steers search towards variables that are common reasons for conflicts

Cons

- Not all the learned clauses are useful
- Memory problems

Lazy Data Structures

Lazy Data Structures

Some things to notice…

- Most features, like clause learning, are based on UP
- Know when you have a unit clause is very important
- Most solvers do not care about $\geq$ binary clauses
- Unnecessary processing of large clauses when only few variables are instantiated

So, let's try a lazy approach…

Lazy Data Structures

With static variable ordering

- One-Watched Literal

With dynamic variable ordering

- Counters (until now, not lazy)
- Head and Tail
- Two-Watched Literal

One-Wathced Literal

One-Watched Literal

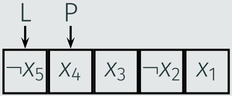
(1) All the literals are free

L	P			
$\neg X_5$	X_4	X_3	$\neg X_2$	X_1

One-Wathced Literal

One-Watched Literal

(1) All the literals are free

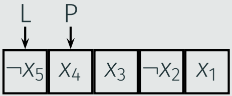


(2) $x_1 = 0, x_2 = 1, x_3 = 0$

One-Wathced Literal

One-Watched Literal

(1) All the literals are free



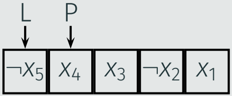
(2) $x_1 = 0, x_2 = 1, x_3 = 0$

(3) $x_4 = 0$

One-Wathced Literal

One-Watched Literal

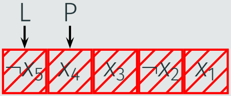
(1) All the literals are free



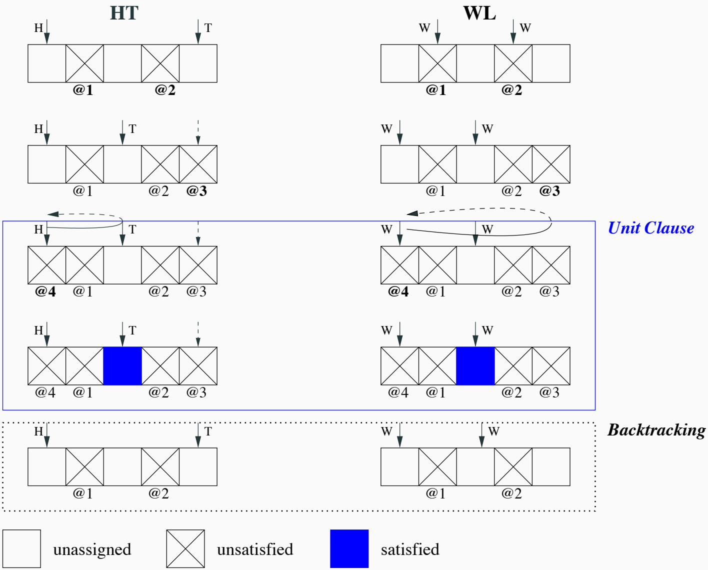
(2) $x_1 = 0, x_2 = 1, x_3 = 0$

(3) $x_4 = 0$

(4) $x_5 = 1$



Head and Tail vs. Two-Watched Literals



Some pros and Cons

With static variable ordering

- One-Watched Literal: Good for static variable selection heuristics

With dynamic variable ordering

- Counters: Keeps track of $\geq$ binary clauses
- Head and Tail (HT): Good for problems with large clauses
- Two-Watched Literal: Like HT but with a cheaper backtracking

MaxSAT

Introduction

In many real-life problems, some potential solutions are acceptable even when they violate some constraints

Max-SAT

Given a Boolean CNF formula ϕ, the Maximum Satisfiability problem (Max-SAT) is the problem of finding a truth assignment that satisfies the maximum number of clauses in ϕ

Weighted Max-SAT

Given a Boolean CNF formula ϕ in which each clause has a weight, the Weighted Max-SAT problem is the problem of finding a truth assignment that maximizes the sum of weights of satisfied clauses in ϕ

Introduction

In many real-life problems, some potential solutions are acceptable even when they violate some constraints

Max-SAT

Given a Boolean CNF formula ϕ, the Maximum Satisfiability problem (Max-SAT) is the problem of finding a truth assignment that satisfies the maximum number of clauses in ϕ

Weighted Max-SAT

Given a Boolean CNF formula ϕ in which each clause has a weight, the Weighted Max-SAT problem is the problem of finding a truth assignment that maximizes the sum of weights of satisfied clauses in ϕ

In many real-life problems, some potential solutions are acceptable even when they violate some constraints

Max-SAT

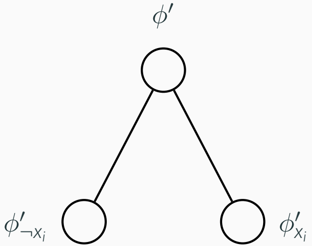
Given a Boolean CNF formula ϕ, the Maximum Satisfiability problem (Max-SAT) is the problem of finding a truth assignment that satisfies the maximum number of clauses in ϕ

Weighted Max-SAT

Given a Boolean CNF formula ϕ in which each clause has a weight, the Weighted Max-SAT problem is the problem of finding a truth assignment that maximizes the sum of weights of satisfied clauses in ϕ

Max-SAT branch and bound algorithms

Given a Max-SAT instance ϕ, we can represent the space of all possible assignments as a binary search tree



Exact Max-SAT solvers explore that search tree in a depth-first manner using a BnB schema

Basic branch and bound Max-SAT algorithm

```
Function Max-SAT (φ : CNF formula, UB : upper bound) : Natural
    if φ = ∅ or φ only contains empty clauses then
        return EmptyClauses(φ)
    end
    LB ← EmptyClauses(φ)
    if LB ≥ UB then
        return UB
    end
    x ← SelectVariable(φ)
    UB ← Min(UB, Max-SAT(φ_¬x, UB))
    return Min(UB, Max-SAT(φ_x, UB))
end
```

Basic branch and bound algorithm

Example

$UB = 10$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$

Basic branch and bound algorithm

Example

$UB = 10$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

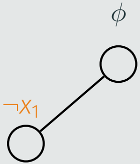
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 10$

$LB = 1$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2 \rightarrow \square$

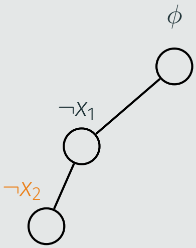
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 2$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3 \rightarrow \square$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2 \rightarrow \square$

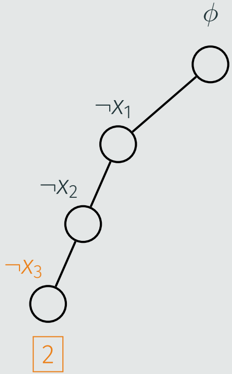
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 1$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2 \rightarrow \square$

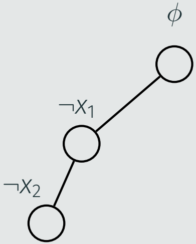
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 2$

$\neg x_1 \lor x_2 \lor \neg x_3$

$\neg x_1 \lor \neg x_2 \lor x_3$

$x_2 \lor x_3$

$\neg x_2 \lor \neg x_3$

$x_1 \lor x_2 \rightarrow \square$

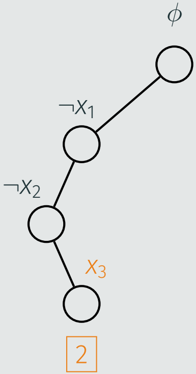
$\neg x_2$

$\neg x_1 \lor \neg x_2$

$x_1 \lor \neg x_2$

$x_1 \lor \neg x_3 \rightarrow \square$

$\neg x_1 \lor x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 1$

$\neg x_1 \lor x_2 \lor \neg x_3$

$\neg x_1 \lor \neg x_2 \lor x_3$

$x_2 \lor x_3$

$\neg x_2 \lor \neg x_3$

$x_1 \lor x_2 \rightarrow \square$

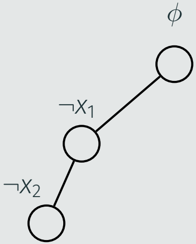
$\neg x_2$

$\neg x_1 \lor \neg x_2$

$x_1 \lor \neg x_2$

$x_1 \lor \neg x_3$

$\neg x_1 \lor x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 0$

$\neg x_1 \lor x_2 \lor \neg x_3$

$\neg x_1 \lor \neg x_2 \lor x_3$

$x_2 \lor x_3$

$\neg x_2 \lor \neg x_3$

$x_1 \lor x_2$

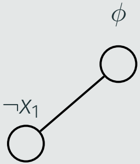
$\neg x_2$

$\neg x_1 \lor \neg x_2$

$x_1 \lor \neg x_2$

$x_1 \lor \neg x_3$

$\neg x_1 \lor x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 2$

$\neg x_1 \lor x_2 \lor \neg x_3$

$\neg x_1 \lor \neg x_2 \lor x_3$

$x_2 \lor x_3$

$\neg x_2 \lor \neg x_3$

$x_1 \lor x_2$

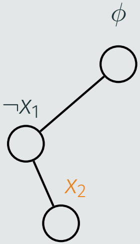
$\neg x_2 \rightarrow \square$

$\neg x_1 \lor \neg x_2$

$x_1 \lor \neg x_2 \rightarrow \square$

$x_1 \lor \neg x_3$

$\neg x_1 \lor x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

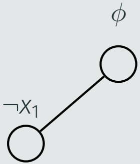
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$

Basic branch and bound algorithm

Example

$UB = 2$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

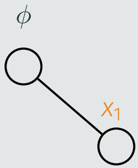
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

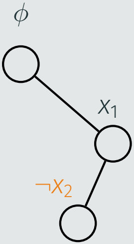
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 2$

$\neg x_1 \lor x_2 \lor \neg x_3$

$\neg x_1 \lor \neg x_2 \lor x_3$

$x_2 \lor x_3 \rightarrow \square$

$\neg x_2 \lor \neg x_3$

$x_1 \lor x_2$

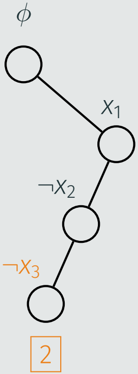
$\neg x_2$

$\neg x_1 \lor \neg x_2$

$x_1 \lor \neg x_2$

$x_1 \lor \neg x_3$

$\neg x_1 \lor x_3 \rightarrow \square$



Basic branch and bound algorithm

Example

$UB = 2$

$LB = 0$

$\neg x_1 \lor x_2 \lor \neg x_3$

$\neg x_1 \lor \neg x_2 \lor x_3$

$x_2 \lor x_3$

$\neg x_2 \lor \neg x_3$

$x_1 \lor x_2$

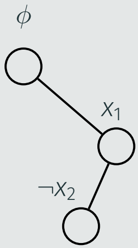
$\neg x_2$

$\neg x_1 \lor \neg x_2$

$x_1 \lor \neg x_2$

$x_1 \lor \neg x_3$

$\neg x_1 \lor x_3$



Basic branch and bound algorithm

Example

$UB = 1$

$LB = 1$

$\neg x_1 \vee x_2 \vee \neg x_3 \rightarrow \square$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

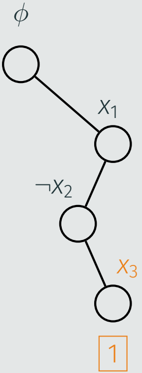
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 1$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

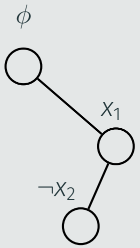
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 1$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

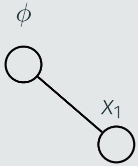
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 1$

$LB = 2$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

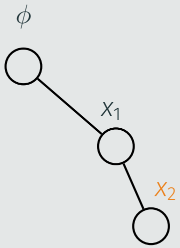
$\neg x_2 \rightarrow \square$

$\neg x_1 \vee \neg x_2 \rightarrow \square$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Basic branch and bound algorithm

Example

$UB = 1$

$LB = 0$

$\neg x_1 \lor x_2 \lor \neg x_3$

$\neg x_1 \lor \neg x_2 \lor x_3$

$x_2 \lor x_3$

$\neg x_2 \lor \neg x_3$

$x_1 \lor x_2$

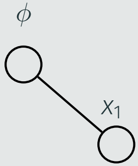
$\neg x_2$

$\neg x_1 \lor \neg x_2$

$x_1 \lor \neg x_2$

$x_1 \lor \neg x_3$

$\neg x_1 \lor x_3$



Basic branch and bound algorithm

Example

$UB = 1$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$

Basic branch and bound algorithm

Example

$UB = 1$

$LB = 0$

$\neg x_1 \vee x_2 \vee \neg x_3$

$\neg x_1 \vee \neg x_2 \vee x_3$

$x_2 \vee x_3$

$\neg x_2 \vee \neg x_3$

$x_1 \vee x_2$

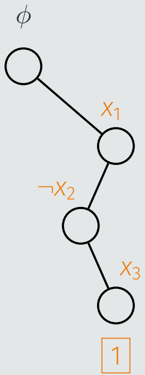
$\neg x_2$

$\neg x_1 \vee \neg x_2$

$x_1 \vee \neg x_2$

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_3$



Improved branch and bound Max-SAT algorithm

Function `Max-SAT` *(ϕ : CNF formula, UB : upper bound)* **: Natural**

```
φ ← SimplifyFormula(φ)

if φ = ∅ or φ only contains empty clauses then
    return EmptyClauses(φ)
end

LB ← EmptyClauses(φ) + Underestimation(φ)
if LB ≥ UB then
    return UB
end

x ← SelectVariable(φ)
UB ← Min(UB, Max-SAT(φ_¬x, UB))
return Min(UB, Max-SAT(φ_x, UB))
end
```

Outline

Introduction

Complete approaches

Improving DPLL

Clause Learning

Lazy Data Structures

MaxSAT

- Max-SAT branch and bound algorithms
- Underestimations
- Inference rules

Lower bounds

$LB \leftarrow \texttt{EmptyClauses}(\phi) + \texttt{Underestimation}(\phi)$

$\texttt{Underestimation}(\phi)$:

- Fast, low overhead
- Accurate

$\texttt{Underestimation}(\phi)$

Based on the detection of disjoint unsatisfiable subsets of clauses in most Max-SAT solvers

Lower bounds

$LB \leftarrow \texttt{EmptyClauses}(\phi) + \texttt{Underestimation}(\phi)$

$\texttt{Underestimation}(\phi)$:

- Fast, low overhead
- Accurate

$\texttt{Underestimation}(\phi)$

Based on the detection of disjoint unsatisfiable subsets of clauses in most Max-SAT solvers

Lower bound examples

Inconsistency counts: $\{x_i, \neg x_i\}$

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

Star rule: $\{\neg x_1, \neg x_2, \ldots, \neg x_k, x_1 \vee x_2 \vee \cdots \vee x_k\}$

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

Lower bound examples

Inconsistency counts: $\{x_i, \neg x_i\}$

$$\underbrace{\boxed{x_1 \wedge \neg x_1}}_{\text{unsatisfiable subset}} \wedge (x_2 \vee x_3) \wedge \neg x_2 \wedge \neg x_3 \wedge (x_1 \vee x_4)$$

Star rule: $\{\neg x_1, \neg x_2, \ldots, \neg x_k, x_1 \vee x_2 \vee \cdots \vee x_k\}$

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

Lower bound examples

Inconsistency counts: $\{x_i, \neg x_i\}$

$$\boxed{x_1 \wedge \neg x_1} \wedge (x_2 \vee x_3) \wedge \neg x_2 \wedge \neg x_3 \wedge (x_1 \vee x_4)$$

unsatisfiable subset

Star rule: $\{\neg x_1, \neg x_2, \ldots, \neg x_k, x_1 \vee x_2 \vee \cdots \vee x_k\}$

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

Lower bound examples

Inconsistency counts: $\{x_i, \neg x_i\}$

$$\underbrace{\boxed{x_1 \land \neg x_1}}_{\text{unsatisfiable subset}} \land (x_2 \lor x_3) \land \neg x_2 \land \neg x_3 \land (x_1 \lor x_4)$$

Star rule: $\{\neg x_1, \neg x_2, \ldots, \neg x_k, x_1 \lor x_2 \lor \cdots \lor x_k\}$

$$\underbrace{\boxed{x_1 \land \neg x_1}}_{\text{unsatisfiable subset}} \land (x_2 \lor x_3) \land \neg x_2 \land \neg x_3 \land (x_1 \lor x_4)$$

Lower bound examples

Inconsistency counts: $\{x_i, \neg x_i\}$

$$\underbrace{\boxed{x_1 \wedge \neg x_1}}_{\text{unsatisfiable subset}} \wedge (x_2 \vee x_3) \wedge \neg x_2 \wedge \neg x_3 \wedge (x_1 \vee x_4)$$

Star rule: $\{\neg x_1, \neg x_2, \ldots, \neg x_k, x_1 \vee x_2 \vee \cdots \vee x_k\}$

$$\underbrace{\boxed{x_1 \wedge \neg x_1}}_{\text{unsatisfiable subset}} \wedge \underbrace{\boxed{(x_2 \vee x_3) \wedge \neg x_2 \wedge \neg x_3}}_{\text{unsatisfiable subset}} \wedge (x_1 \vee x_4)$$

Lower bound UP

Number of disjoint unsatisfiable subsets of clauses detected by unit propagation

Example

$$x_1$$
$$\neg x_1 \vee x_2$$
$$\neg x_1 \vee \neg x_2$$
$$x_1 \vee x_3$$

Lower bound UP

Number of disjoint unsatisfiable subsets of clauses detected by unit propagation

Example

$$\begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array} \quad \Rightarrow \quad \begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array}$$

Lower bound UP

Number of disjoint unsatisfiable subsets of clauses detected by unit propagation

Example

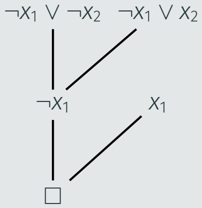
$$\begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array} \Rightarrow \begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array} \Rightarrow \square$$

Lower bound UP

Number of disjoint unsatisfiable subsets of clauses detected by unit propagation

Example

$$\begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array} \quad \Rightarrow \quad \begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array} \quad \Rightarrow \quad \square$$

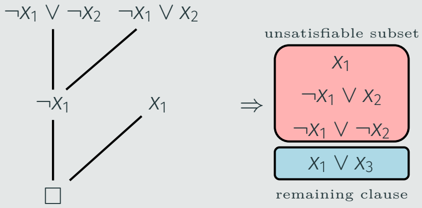


Lower bound UP

Number of disjoint unsatisfiable subsets of clauses detected by unit propagation

Example

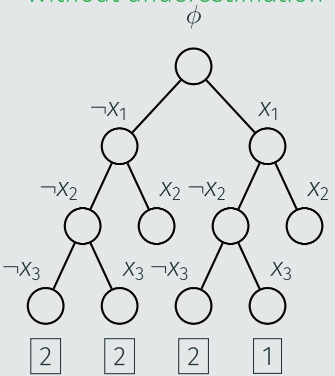
$$
\begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array}
\quad \Rightarrow \quad
\begin{array}{c} x_1 \\ \neg x_1 \vee x_2 \\ \neg x_1 \vee \neg x_2 \\ x_1 \vee x_3 \end{array}
\quad \Rightarrow \quad \square
$$



Lower bound UP

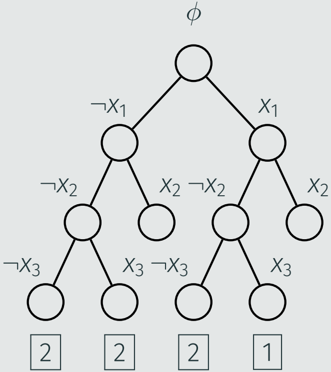
Example

Without underestimation

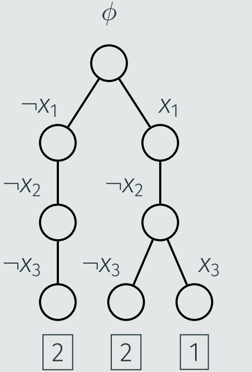


Example

Without underestimation



With lower bound UP



Lower bound UP + failed literals

Example

$\neg x_1 \vee \neg x_2$

$\neg x_1 \vee x_2$

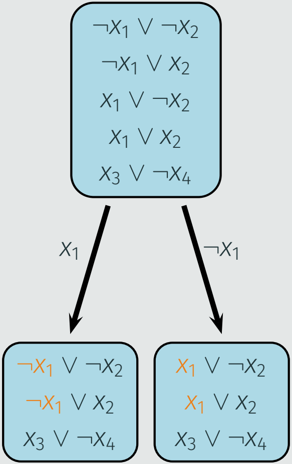
$x_1 \vee \neg x_2$

$x_1 \vee x_2$

$x_3 \vee \neg x_4$

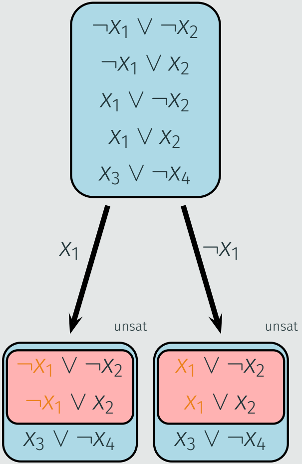
Lower bound UP + failed literals

Example



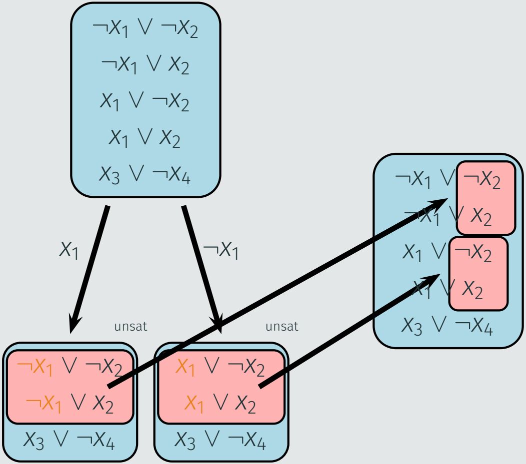
Lower bound UP + failed literals

Example



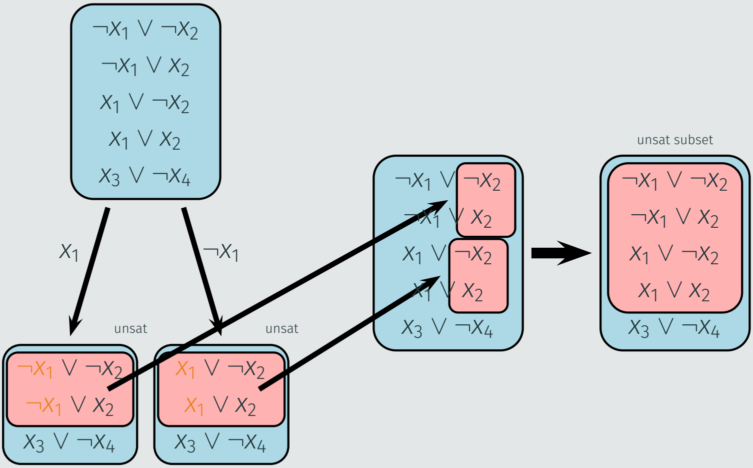
Lower bound UP + failed literals

Example



Lower bound UP + failed literals

Example



Outline

Introduction

Complete approaches

Improving DPLL

Clause Learning

Lazy Data Structures

MaxSAT

- Max-SAT branch and bound algorithms
- Underestimations
- Inference rules

Inference rules

Max-SAT resolution rule

This rule provides a complete calculus for Max-SAT

$$
\begin{array}{l}
x \vee a_1 \vee \cdots \vee a_s \\
\bar{x} \vee b_1 \vee \cdots \vee b_t \\
\hline\hline
a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\
x \vee a_1 \vee \cdots \vee a_s \vee \bar{b}_1 \\
x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \bar{b}_2 \\
\cdots \\
x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \bar{b}_t \\
\bar{x} \vee b_1 \vee \cdots \vee b_t \vee \bar{a}_1 \\
\bar{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \bar{a}_2 \\
\cdots \\
\bar{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \bar{a}_s
\end{array}
$$

- Max-SAT solvers use refinements of the Max-SAT resolution rule
- Inference rules substitute the premises by the conclusions

Inference rules

- Inference without underestimation (Toolbar)
- Inference guided by UP (MaxSatz)
- Limited resolution guided by UP (MiniMaxSat)

Inference without underestimation (Toolbar)

Chain resolution

$$\left\{ \begin{array}{l} (l_1, w_1), \\ (\bar{l}_i \vee l_{i+1}, w_{i+1})_{1 \le i < k}, \\ (\bar{l}_k, w_{k+1}) \end{array} \right\} \Longrightarrow \left\{ \begin{array}{l} (l_i, m_i - m_{i+1})_{1 \le i \le k}, \\ (\bar{l}_i \vee l_{i+1}, w_{i+1} - m_{i+1})_{1 \le i < k}, \\ (l_i \vee \bar{l}_{i+1}, m_{i+1})_{1 \le i < k}, \\ (\bar{l}_k, w_{k+1} - m_{k+1}), \\ (\square, m_{k+1}) \end{array} \right\}$$

Cycle resolution (restricted to $k = 3$)

$$\left\{ \begin{array}{l} (\bar{l}_i \vee l_{i+1}, w_i)_{1 \le i < k}, \\ (\bar{l}_1 \vee \bar{l}_k, w_k) \end{array} \right\} \Longrightarrow \left\{ \begin{array}{l} (\bar{l}_1 \vee l_i, m_{i-1} - m_i)_{2 \le i \le k}, \\ (\bar{l}_i \vee l_{i+1}, w_i - m_i)_{2 \le i < k}, \\ (\bar{l}_1 \vee l_i \vee \bar{l}_{i+1}, m_i)_{2 \le i < k}, \\ (l_1 \vee \bar{l}_i \vee l_{i+1}, m_i)_{2 \le i < k}, \\ (\bar{l}_1 \vee \bar{l}_k, w_k - m_k), \\ (\bar{l}_1, m_k) \end{array} \right\}$$

Inference without underestimation (Toolbar)

Chain resolution

$$\left\{\begin{array}{l}(l_1, w_1),\\ (\bar{l}_i \vee l_{i+1}, w_{i+1})_{1\le i<k},\\ (\bar{l}_k, w_{k+1})\end{array}\right\} \Longrightarrow \left\{\begin{array}{l}(l_i, m_i - m_{i+1})_{1\le i\le k},\\ (\bar{l}_i \vee l_{i+1}, w_{i+1} - m_{i+1})_{1\le i<k},\\ (l_i \vee \bar{l}_{i+1}, m_{i+1})_{1\le i<k},\\ (\bar{l}_k, w_{k+1} - m_{k+1}),\\ (\square, m_{k+1})\end{array}\right\}$$

Cycle resolution (restricted to $k = 3$)

$$\left\{\begin{array}{l}(\bar{l}_i \vee l_{i+1}, w_i)_{1\le i<k},\\ (\bar{l}_1 \vee \bar{l}_k, w_k)\end{array}\right\} \Longrightarrow \left\{\begin{array}{l}(\bar{l}_1 \vee l_i, m_{i-1} - m_i)_{2\le i\le k},\\ (\bar{l}_i \vee l_{i+1}, w_i - m_i)_{2\le i<k},\\ (\bar{l}_1 \vee l_i \vee \bar{l}_{i+1}, m_i)_{2\le i<k},\\ (l_1 \vee \bar{l}_i \vee l_{i+1}, m_i)_{2\le i<k},\\ (\bar{l}_1 \vee \bar{l}_k, w_k - m_k),\\ (\bar{l}_1, m_k)\end{array}\right\}$$

Inference guided by UP (MaxSatz)

Inference rules in MaxSatz

$$\frac{l_1 \quad \neg l_1}{\Box}$$

$$\frac{l_1 \lor l_2 \quad \neg l_1 \lor l_2}{l_2}$$

$$\frac{l_1 \quad l_2 \quad \neg l_1 \lor \neg l_2}{\Box \quad l_1 \lor l_2}$$

$$\frac{l_1 \quad \neg l_1 \lor l_2 \quad \neg l_1 \lor l_3 \quad \neg l_2 \lor \neg l_3}{\Box \quad l_1 \lor \neg l_2 \lor \neg l_3 \quad \neg l_1 \lor l_2 \lor l_3}$$

$$\begin{array}{c} l_1 \\ \neg l_1 \lor l_2 \\ \neg l_2 \lor l_3 \\ \vdots \\ \neg l_k \lor l_{k+1} \\ \neg l_{k+1} \end{array} \Rightarrow \begin{array}{c} \Box \\ l_1 \lor \neg l_2 \\ l_2 \lor \neg l_3 \\ \vdots \\ l_k \lor \neg l_{k+1} \end{array}$$

$$\begin{array}{c} l_1 \\ \neg l_1 \lor l_2 \\ \neg l_2 \lor l_3 \\ \vdots \\ \neg l_k \lor l_{k+1} \\ \neg l_{k+1} \lor l_{k+2} \\ \neg l_{k+1} \lor l_{k+3} \\ \neg l_{k+2} \lor \neg l_{k+3} \end{array} \Rightarrow \begin{array}{c} \Box \\ l_1 \lor \neg l_2 \\ l_2 \lor \neg l_3 \\ \vdots \\ l_k \lor \neg l_{k+1} \\ l_{k+1} \lor \neg l_{k+2} \lor \neg l_{k+3} \\ l_{k+1} \lor l_{k+2} \lor l_{k+3} \end{array}$$

Limited resolution guided by UP (MiniMaxSat)

Once UP derives a contradiction, it builds a refutation:

if all the resolvents have size less than 4 **then** apply

Max-SAT resolution rule

$$\frac{\begin{array}{l} x \vee a_1 \vee \cdots \vee a_s \\ \bar{x} \vee b_1 \vee \cdots \vee b_t \end{array}}{\begin{array}{l} a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\ x \vee a_1 \vee \cdots \vee a_s \vee \bar{b}_1 \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \bar{b}_2 \\ \cdots \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \bar{b}_t \\ \bar{x} \vee b_1 \vee \cdots \vee b_t \vee \bar{a}_1 \\ \bar{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \bar{a}_2 \\ \cdots \\ \bar{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \bar{a}_s \end{array}}$$

else

increment the underestimation

Advanced Programming in Artificial Intelligence

Josep Argelich
Degree in Computer Engineering