Constraint Solving

René Thiemann and Fabian Mitterwallner

based on a previous course by Aart Middeldorp

Outline

SAT Solving

- conflict graph is used to compute backjump clauses
- two new inference rules: learn (a backjump clause) and restart

Theorem (Cook-Levin)

SAT is NP-complete

SAT Variation

k-SAT: every clause has (at most) k literals

Theorem

- 3SAT is NP-complete
- 2SAT is solvable in polynomial time

Planar 3SAT

instance is 3SAT formula φ whose incidence graph is planar

- φ with clauses $\mathcal{C} = \{C_1, \dots, C_m\}$ over variables $\mathcal{V} = \{x_1, \dots, x_n\}$
- bipartite graph $(\mathcal{C} \cup \mathcal{V}, \mathcal{E})$ with $\mathcal{E}$ containing edge $C_i - x_j$ if and only if C_i contains x_j or $\neg x_j$

Theorem (Lichtenstein 1982)

planar 3SAT is NP-complete

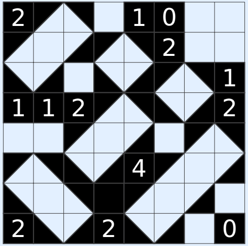
Remark

planar 3SAT is often used in reductions to show NP-hardness of particular problems

Outline

1. Summary of Previous Lecture
2. **Example SAT Reduction: Shakashaka**
3. Maximal Satisfiability
4. Incremental SAT Solving
5. Further Reading

Shakashaka



- fill grid with triangles ◣ ◥ ◤ ◢ to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

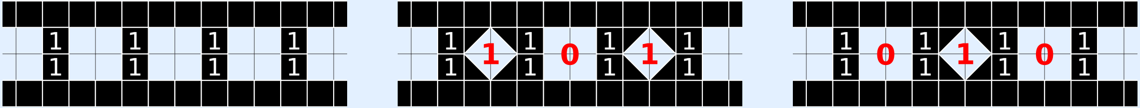
Theorem

Shakashaka is NP-hard

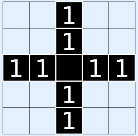
Proof

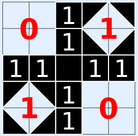
reduction from planar 3SAT

Wires

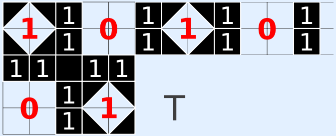


Variable Gadgets





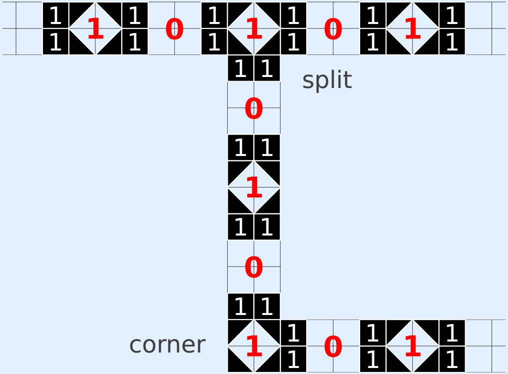
F



T

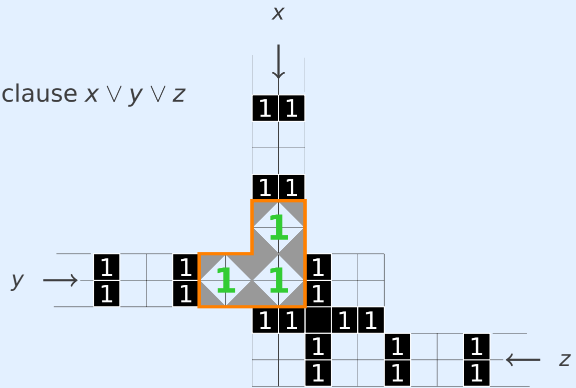
two possible solutions $\approx$ two possible valuations

Split and Corner Gadgets



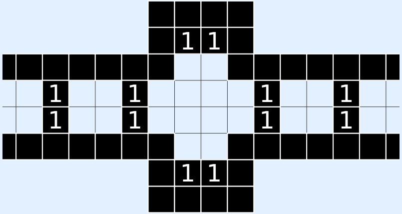
Clause Gadget

clause $x \lor y \lor z$

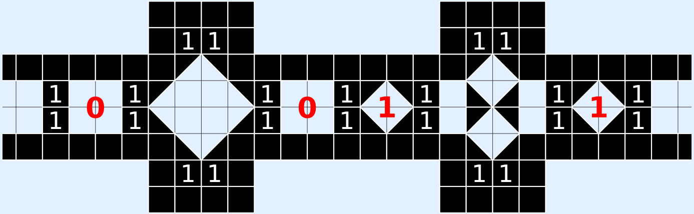


x	y	z		
0	0	0	✗	no rectangle
0	0	1	✓	
0	1	0	✓	
0	1	1	✓	
1	0	0	✓	
1	0	1	✓	
1	1	0	✓	
1	1	1	✓	

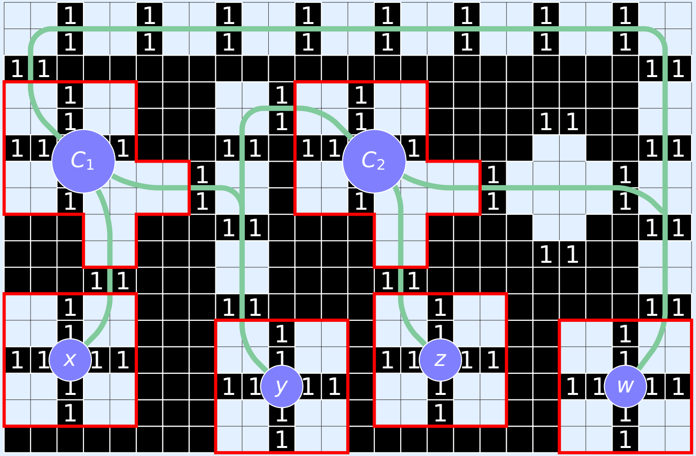
Parity Gadget



to change position of wires



Example



- 4 variables $x\ y\ z\ w$
- 2 clauses $C_1\ C_2$

$$C_1 = \{x, \neg y, w\}$$

$$C_2 = \{y, \neg z, \neg w\}$$

Outline

MaxSAT

input: propositional CNF φ

output: valuation v that maximizes number of satisfied clauses in φ

Example

- CNF $(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee r) \wedge \neg p \wedge (\neg p \vee \neg r \vee q)$ is unsatisfiable
- $v(p) = v(q) = v(r) = \mathsf{T}$ satisfies 5 out of 6 clauses

Variation

- hard clauses that must be satisfied
- soft clauses that are desirable to be satisfied
- weights for soft clauses
- goal: maximize sum of weights of soft clauses while satisfying all hard clauses

Branch and Bound — Notation

- φ_x denotes formula φ with all occurrences of x replaced by T
- $\varphi_{\overline{x}}$ denotes formula φ with all occurrences of x replaced by F
- function simp(φ)
  - replaces $\neg$T by F and $\neg$F by T
  - removes all clauses which contain T
  - removes F from remaining clauses
- #empty(φ) denotes number of empty clauses in φ
- CNF φ is presented as list of clauses

Branch & Bound — Algorithm

function BnB(φ, b) -- initial call: b is number $|\varphi|$ of clauses in φ

$\varphi \leftarrow \mathsf{simp}(\varphi)$

if φ contains only empty clauses then return #empty(φ)

if #empty(φ) $\geqslant b$ then return b

x $\leftarrow$ select variable in φ

return $\min(b, \mathsf{BnB}(\varphi_x, b), \mathsf{BnB}(\varphi_{\overline{x}}, \mathsf{BnB}(\varphi_x, b)))$

Remarks

- BnB($\varphi, |\varphi|$) returns minimum number of unsatisfied clauses (minUNSAT(φ))
- maxSAT answer is $|\varphi| - \mathsf{BnB}(\varphi, |\varphi|)$
- #empty(φ) denotes number of clauses falsified by current valuation

Example

- $\varphi = x,\ \neg x \vee y,\ z \vee \neg y,\ x \vee z,\ x \vee y,\ \neg y$

  $\text{simp}(\varphi) = \varphi$ $\qquad$ $\#\text{empty} = 0$
- $\varphi_x = \text{T},\ \neg\text{T} \vee y,\ z \vee \neg y,\ \text{T} \vee z,\ \text{T} \vee y,\ \neg y$

  $\text{simp}(\varphi_x) = y,\ z \vee \neg y,\ \neg y$ $\qquad$ $\#\text{empty} = 0$
- $\varphi_{xy} = \text{T},\ z \vee \neg\text{T},\ \neg\text{T}$

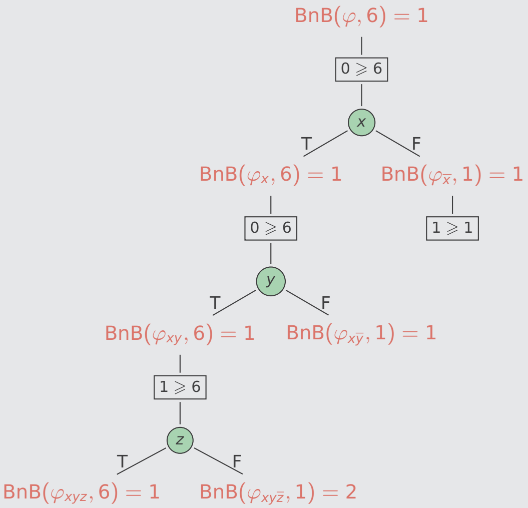
  $\text{simp}(\varphi_{xy}) = z,\ \square$ $\qquad$ $\#\text{empty} = 1$
- $\varphi_{xyz} = \text{T},\ \square$

  $\text{simp}(\varphi_{xyz}) = \square$ $\qquad$ $\#\text{empty} = 1$
- $\varphi_{xy\bar{z}} = \text{F},\ \square$

  $\text{simp}(\varphi_{xy\bar{z}}) = \square,\ \square$ $\qquad$ $\#\text{empty} = 2$
- $\varphi_{x\bar{y}} = \text{F},\ z \vee \neg\text{F},\ \neg\text{F}$

  $\text{simp}(\varphi_{x\bar{y}}) = \square$ $\qquad$ $\#\text{empty} = 1$
- $\varphi_{\bar{x}} = \text{F},\ \neg\text{F} \vee y,\ z \vee \neg y,\ \text{F} \vee z,\ \text{F} \vee y,\ \neg y$

  $\text{simp}(\varphi_{\bar{x}}) = \square,\ z \vee \neg y,\ z,\ y,\ \neg y$ $\qquad$ $\#\text{empty} = 1$
- $\text{maxSAT}(\varphi) = 6 - \text{BnB}(\varphi, 6) = 5$

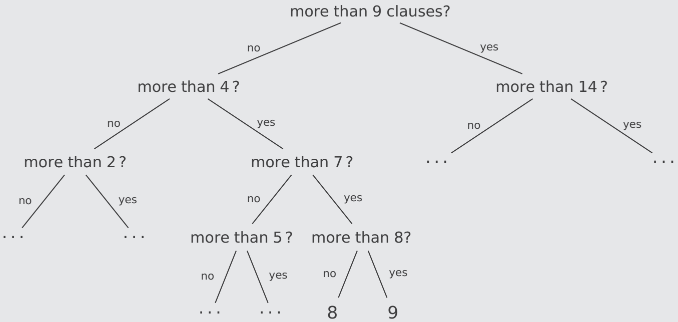


Binary Search — Idea

repeatedly call SAT solver in binary search fashion and return minUNSAT(φ)

Example

consider formula with 18 clauses; can we satisfy ...



Binary Search

function BinarySearch($\{C_1, \dots, C_m\}$)

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$ -- $b_1, \dots, b_m$ are fresh variables

return search($\varphi, 0, m$)

function search(φ, L, U)

if $L \geqslant U$ then return U

$M = \lfloor \frac{L+U}{2} \rfloor$

if SAT($\varphi \wedge$ CNF($b_1 + \cdots + b_m \leqslant M$)) then return search($\varphi, L, M$) cardinality constraint

else return search($\varphi, M+1, U$)

Theorem

BinarySearch(φ) = minUNSAT(φ)

Example

$$\varphi = \left\{ \begin{array}{llll} 6 \vee 2 \vee b_1 & \neg 6 \vee 2 \vee b_2 & \neg 2 \vee 1 \vee b_3 & \neg 1 \vee b_4 \\ \neg 6 \vee 8 \vee b_5 & 6 \vee \neg 8 \vee b_6 & 2 \vee 4 \vee b_7 & \neg 4 \vee 5 \vee b_8 \\ 7 \vee 5 \vee b_9 & \neg 7 \vee 5 \vee b_{10} & \neg 3 \vee b_{11} & \neg 5 \vee 3 \vee b_{12} \end{array} \right\}$$

- $L = 0$, $U = 12$, $M = 6$ $\quad$ SAT($\varphi \wedge \mathrm{CNF}(b_1 + \cdots + b_{12} \leqslant 6)$) ? ✓
- $L = 0$, $U = 6$, $M = 3$ $\quad$ SAT($\varphi \wedge \mathrm{CNF}(b_1 + \cdots + b_{12} \leqslant 3)$) ? ✓
- $L = 0$, $U = 3$, $M = 1$ $\quad$ SAT($\varphi \wedge \mathrm{CNF}(b_1 + \cdots + b_{12} \leqslant 1)$) ? ×
- $L = 2$, $U = 3$, $M = 2$ $\quad$ SAT($\varphi \wedge \mathrm{CNF}(b_1 + \cdots + b_{12} \leqslant 2)$) ? ✓
- $L = 2$, $U = 2$ $\quad$ $\mathrm{minUNSAT}(\varphi) = 2 \implies \mathrm{maxSAT}(\varphi) = 10$

MaxSAT Competition

Further Information

- MaxSAT Evaluation 2023

Outline

Motivation

- applications such as MaxSAT often require incremental interface
  - determine satisfiability of clauses C_1
  - slightly change C_1 to C_2 and again determine satisfiability
  - ...
- aims
  - do not restart SAT solver in every iteration
  - reuse learned clauses, keep knowledge of solver (decision heuristics, ...)
- challenge
  - keep learned clauses, even if some clauses are removed from input

Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: assumption literals (also called clause selectors)
  - create new assumption literal x_c for each clause c that might be activated or deactivated
  - change every such clause c in the CNF by $\{\neg x_c\} \cup c$; CNF is $\bigcup_i C_i$
  - run DPLL algorithm for C_1 with initial decisions x_c^d for each $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of C_1
  - for switching from clauses C_1 to C_2 perform two steps
    - undo decisions x_c^d for all $c \in C_1 \setminus C_2$ (deactivate) and
    - add decisions x_c^d for all $c \in C_2 \setminus C_1$ (activate)
  - then continue with DPLL algorithm as in previous step, continue with $C_3, \ldots$

Observation

- since the set of clauses stays identical, learned clauses stay valid
- learned clauses that contain clause selectors can only be applied if these clauses are currently activated; other learned clauses can be used for all runs

Outline

Kröning and Strichmann

- Sections 2.2 and 2.4

Further Reading

- Erik D. Demaine, Yoshio Okamata, Ryuhei Uehara, and Yushi Uno
  Computational Complexity and an Integer Programming Model of Shakashaka
  Proc. 25th Canadian Conference on Computational Geometry, 2013

Motivation for extending SAT to first-order theories

predicates instead of propositional variables

Examples

- equalities and disequalities over the reals

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$

- boolean combination of linear-arithmethic predicates

$$(x_1 + 2x_3 < 5) \vee \neg(x_3 \leqslant 1) \wedge (x_1 \geqslant x_3)$$

- formula over arrays

$$(i = j \wedge a[j] = 1) \wedge \neg(a[i] = 1)$$

Important Concepts

- assumption literal (clause selector)
- binary search
- branch and bound
- hard and soft clause
- incremental SAT solving
- maximal satisfiability
- minUNSAT
- Shakashaka gadget