Algorithm Design & Analysis (CS3383)[1]

Unit 2: Backtracking and SAT

Rasoul Shahsavarifar

February 1, 2019

[1]Thanks to Dr. Ptricia Evans and Dr. David Bremner for sharing the teaching stuffs

Outline[2]

Combinatorial Search

- Backtracking
- SAT
- Tractable kinds of SAT

---

[2] Reading:

- Main textbook (DPV), Divide and conquer algorithms, Chapter 9 mainly 9.1.1 and 9.1.2.
- Backtracking algorithms from Jeff Ericson's Algorithm page `http://jeffe.cs.illinois.edu/teaching/algorithms/book/02-backtracking.pdf`

Contents

The **Backtracking** approach is based on trying a promising direction, developing and testing the possible solution, and then backing up to try related possibilities if the solution does not work.

Backtracking is essentially performing a Depth-First Search of the solution space. Many problems can be encoded as a boolean formula, where a solution to the problem corresponds to a satisfying truth assignment for the formula.

The general problem, SATISFIABILITY, is NP-complete.

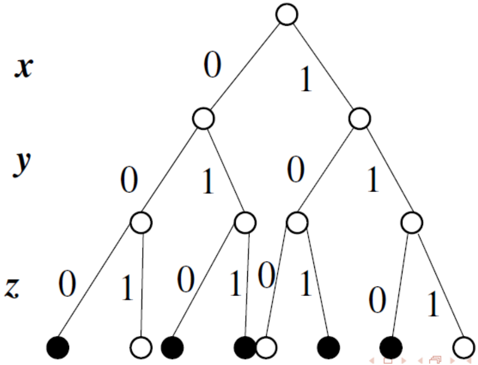
A boolean formula can be converted to Conjunctive Normal Form (a conjunction of disjunctions), *e.g.*

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

The CNF-SAT problem is also NP-complete.

CNF-SAT Example

$$(x \vee \overline{y}) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (\overline{y} \vee z) \wedge (x \vee z)$$



N-queens

Problem Description

Given an $n \times n$ chess board, can you place n queens so that no two are in the same row, column, or diagonal.

Examples

	Q		
			Q
Q			
		Q	

$\checkmark$

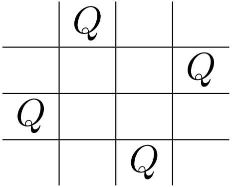
	Q		
			Q
	Q		
		Q	

no

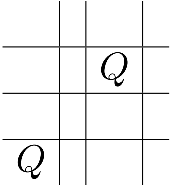
- We can eliminate the problem case on the right immediately

Representing Chessboards

- We only care about cases where there is 1 queen per column
- Represent a $n \times n$ board as an array of n integers, meaning which row.
- 0 for not chosen yet.

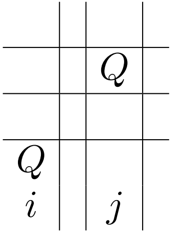


$(2, 4, 1, 3)$



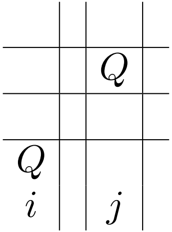
$(4, 0, 2, 0)$

Detecting collisions

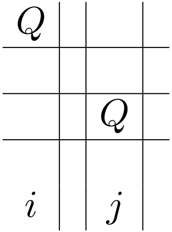


$$Q[j] - Q[i] = j - i$$

Detecting collisions

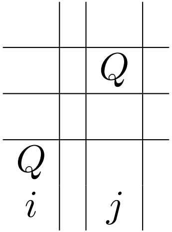


$$Q[j] - Q[i] = j - i$$

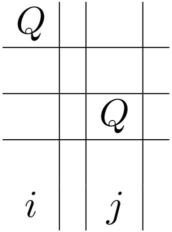


$$Q[j] - Q[i] = i - j$$

Detecting collisions



$$Q[j] - Q[i] = j - i$$



$$Q[j] - Q[i] = i - j$$

- And one more (easy) case

Backtracking Requirements

1. A representation for partial solutions

Backtracking Requirements

1. A representation for partial solutions
2. A procedure to expand a problem into smaller subproblems

Backtracking Requirements

1. A representation for partial solutions
2. A procedure to expand a problem into smaller subproblems
3. A test for partial solutions that returns

Backtracking Requirements

1. A representation for partial solutions
2. A procedure to expand a problem into smaller subproblems
3. A test for partial solutions that returns SUCCESS if the solution is complete

Backtracking Requirements

1. A representation for partial solutions
2. A procedure to expand a problem into smaller subproblems
3. A test for partial solutions that returns
   SUCCESS if the solution is complete
   FAILURE if there is no way to complete

Backtracking Requirements

1. A representation for partial solutions
2. A procedure to expand a problem into smaller subproblems
3. A test for partial solutions that returns

   SUCCESS if the solution is complete
   FAILURE if there is no way to complete
   UNKNOWN if neither of the above can be quickly determined.

Generic Backtracking

```
function BACKTRACK(P_0)
    S ← { P_0 }
    while ! empty(S) do
        P ← S.dequeue()
        for R ∈ expand(P) do
            switch test(R) do
                case SUCCESS
                    return SUCCESS
                case UNKNOWN
                    S.enqueue(R)
        end for
    end while
    return FAIL
```

Backtracking for N-Queens: Framework

representation $Q[1...n]$ where $Q[i]$ is row chosen, or 0 for none.

Backtracking for N-Queens: Framework

representation $Q[1...n]$ where $Q[i]$ is row chosen, or 0 for none.

expand For some $Q[i] = 0$, try $Q[i] = 1...n$

Backtracking for N-Queens: Framework

representation $Q[1...n]$ where $Q[i]$ is row chosen, or 0 for none.

expand For some $Q[i] = 0$, try $Q[i] = 1...n$

```
def test(Q):
    default ← SUCCESS
    for i ∈ 1 ... n − 1:
        if Q[i] = 0:
            default ← UNKNOWN
        else:
            for j ∈ 1 ... i − 1:
                if Q[i] − Q[j] ∈ { 0, i − j, j − i }:
                    return FAIL
    return default
```

N-Queen is an NP-Comlete problem!

Verifiable?

- NP-Complete problems are verifiable by some P problems!
- How to verify the N-queen problem? (board)

N-Queen is an NP-Comlete problem!

Verifiable?

- NP-Complete problems are verifiable by some P problems!
- How to verify the N-queen problem? (board)

Sudoku: another NP-Complete

- `https://en.wikipedia.org/wiki/Sudoku_solving_algorithms`
- Exercise: write a pseudocode for Sudoku.
  - How to verify the Sudoku by a P algorithm?

Backtracking for subset sum

Subset Sum

Given $X \subset \mathbb{R}+$, T

Decide Is there a subset of X that sums to T

Reduction

Reduce the Subset Sum to linear combination problem. (board)

Backtracking for subset sum

Subset Sum

Given $X \subset \mathbb{R}+$, T

Decide Is there a subset of X that sums to T

Reduction

Reduce the Subset Sum to linear combination problem. (board)

Branching

Backtracking for subset sum

Subset Sum

Given $X \subset \mathbb{R}+$, T

Decide Is there a subset of X that sums to T

Reduction

Reduce the Subset Sum to linear combination problem. (board)

Branching

- If (X, T) is feasible for some Z, for all $y \in X$, either the solution includes y or not.

Backtracking for SubsetSum

```
function SubsetSum(X,T)
    if T = 0 then
        return true
    elseif T < 0 or X = ∅
        return false
    end
    (y, X′) ← pop(X)
    return SubsetSum(X′, T − y)
        or SubsetSum(X′, T)
end
```

Contents

The SAT Problem

Conjunctive Normal Form (CNF)

Variables $\{ x_1 ... x_n \}$

Literals $L = \{ x_i, \bar{x}_i \mid \text{variable } x_i \}$

Clauses $\{ z_1, ..., z_k \} \subset L$

The SAT Problem

Conjunctive Normal Form (CNF)

Variables $\{ x_1 ... x_n \}$

Literals $L = \{ x_i, \bar{x}_i \mid \text{variable } x_i \}$

Clauses $\{ z_1, ..., z_k \} \subset L$

Propositional Satisfiability (SAT)

Instance Set of clauses $\mathcal{C}$

Question Is there an assignment of 0, 1 to every variable such that each clause has at least one true literal?

SAT Example

$$\{\{1,2,3\},\{-1,-2,-3\}\} = \{\{x_1,x_2,x_3\},\{\bar{x}_1,\bar{x}_2,\bar{x}_3\}\}$$
$$=$$
(A)

Truth Table

x_1	x_2	x_3	A
0	0	0	
0	0	1	
0	1	0	
	$\vdots$		

SAT Example

$$\{\{1,2,3\},\{-1,-2,-3\}\} = \{\{x_1,x_2,x_3\},\{\bar{x}_1,\bar{x}_2,\bar{x}_3\}\}$$
$$= (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

(A)

Truth Table

x_1	x_2	x_3	A
0	0	0	
0	0	1	
0	1	0	
	$\vdots$		

Backtracking for SAT

representation (reduced) clauses

test if empty clause, return FAIL. If no clauses, return SUCCESS. Otherwise return UNKNOWN

expand $P_0 =$ reduce$(P, j, 0)$, $P_1 =$ reduce$(P, j, 1)$ for some j.

Backtracking SAT Example II

$(w \vee x \vee y \vee z), (w \vee \overline{x}), (x \vee \overline{y}), (y \vee \overline{z}), (z \vee \overline{w}), (\overline{w} \vee \overline{z})$

$w = 0$

$(x \vee y \vee z), (\overline{x}), (x \vee \overline{y}), (y \vee \overline{z})$

Backtracking SAT Example II

$(w \lor x \lor y \lor z), (w \lor \overline{x}), (x \lor \overline{y}), (y \lor \overline{z}), (z \lor \overline{w}), (\overline{w} \lor \overline{z})$

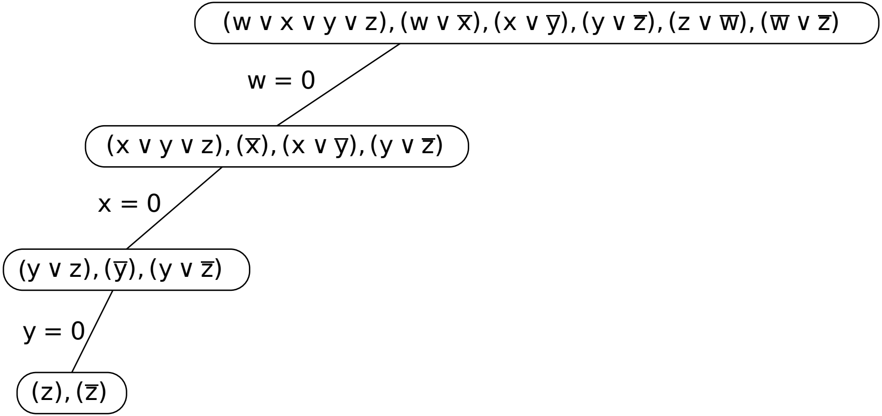
$w = 0$

$(x \lor y \lor z), (\overline{x}), (x \lor \overline{y}), (y \lor \overline{z})$

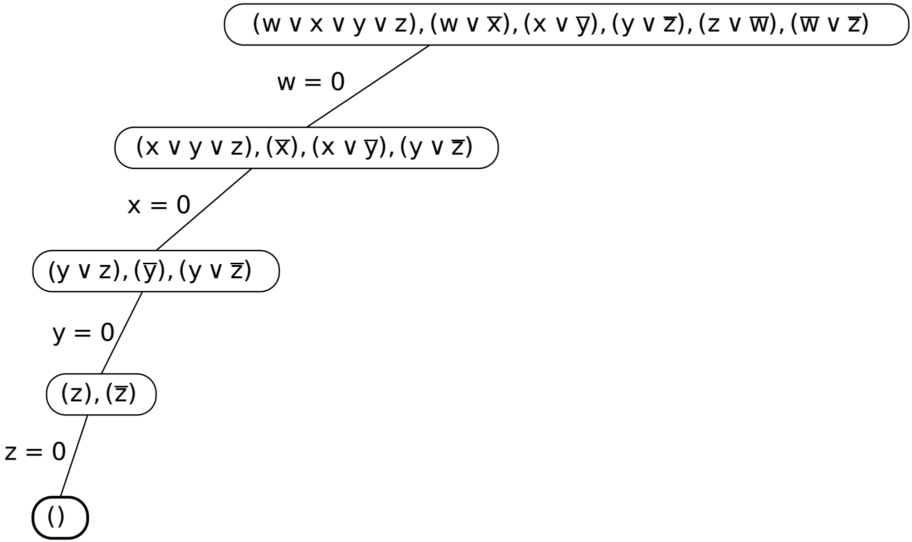
$x = 0$

$(y \lor z), (\overline{y}), (y \lor \overline{z})$

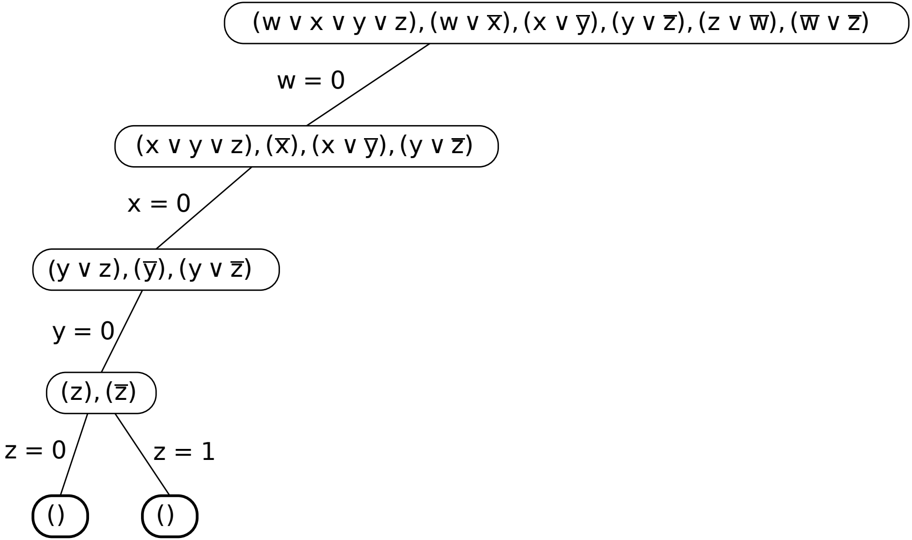
Backtracking SAT Example II



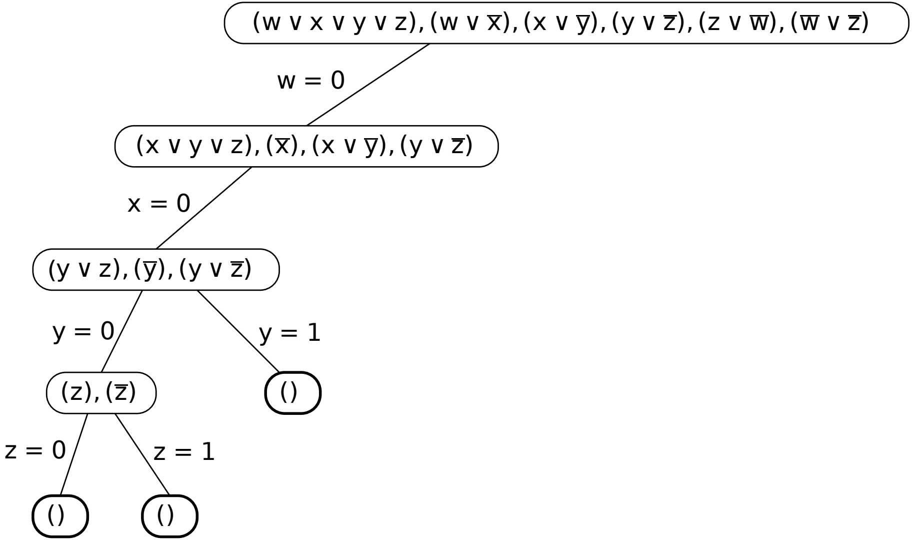
Backtracking SAT Example II



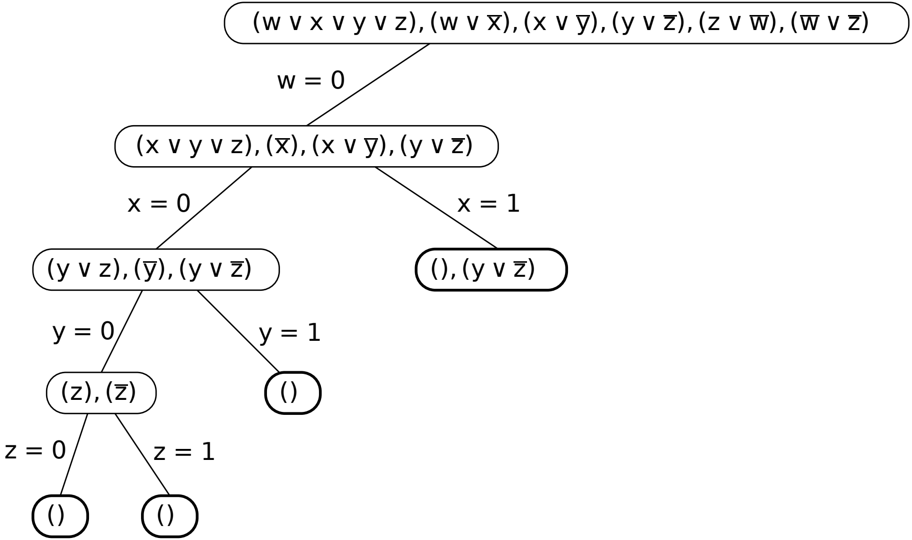
Backtracking SAT Example II



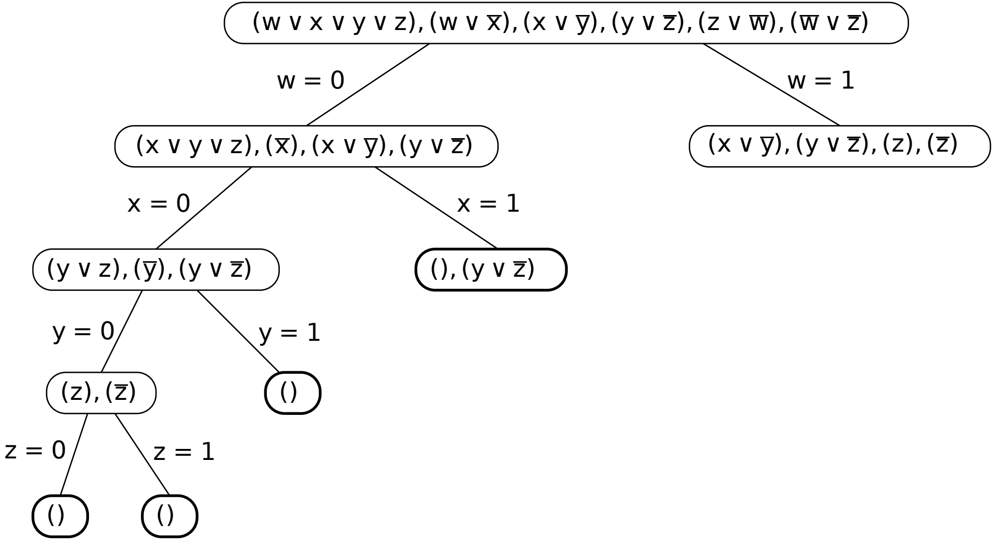
Backtracking SAT Example II



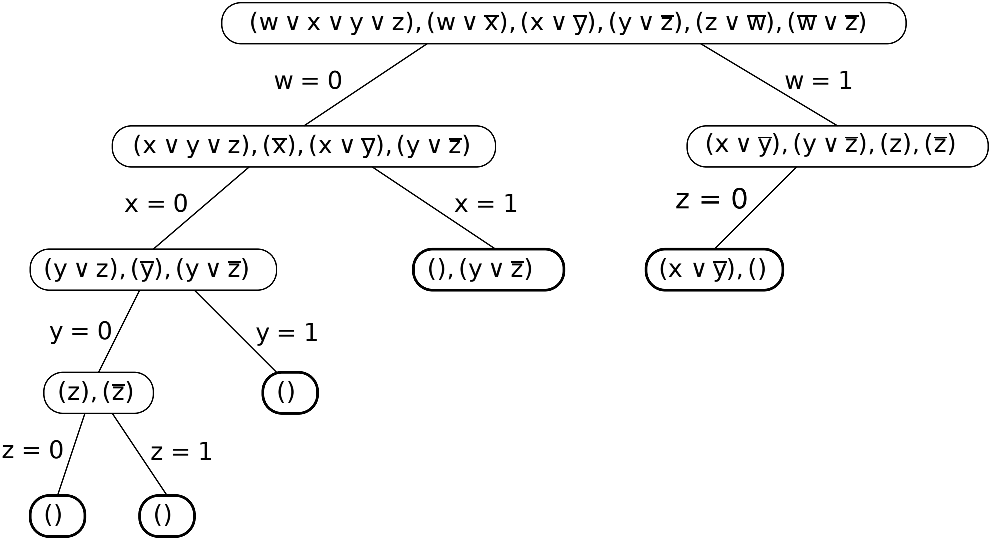
Backtracking SAT Example II



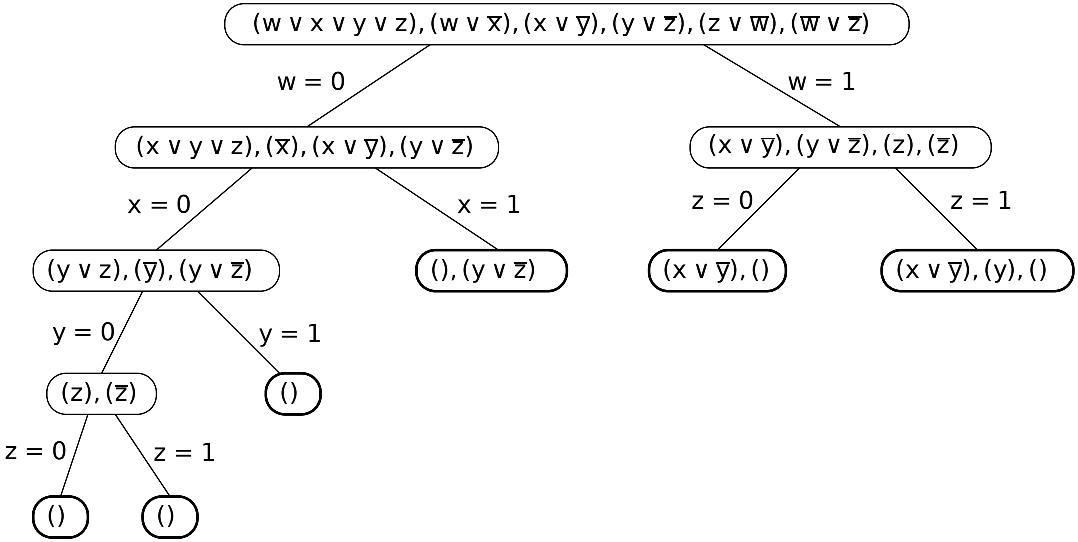
Backtracking SAT Example II



Backtracking SAT Example II



Backtracking SAT Example II



Contents

2SAT

- In 2SAT problem every clause has at most 2 elements

2SAT

- In 2SAT problem every clause has at most 2 elements
- 2SAT is solvable in polynomial time, but not quite trivially.

2SAT

- In 2SAT problem every clause has at most 2 elements
- 2SAT is solvable in polynomial time, but not quite trivially.
- Greedy fails on

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_4 \vee x_5) \wedge (\bar{x}_4 \vee x_6)$$

2SAT

- In 2SAT problem every clause has at most 2 elements
- 2SAT is solvable in polynomial time, but not quite trivially.
- Greedy fails on

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_4 \vee x_5) \wedge (\bar{x}_4 \vee x_6)$$

- to maximize number of clauses satisfied, choose $x_1 \leftarrow 1$, $x_4 \leftarrow 0$

2SAT

- In 2SAT problem every clause has at most 2 elements
- 2SAT is solvable in polynomial time, but not quite trivially.
- Greedy fails on

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_4 \vee x_5) \wedge (\bar{x}_4 \vee x_6)$$

- to maximize number of clauses satisfied, choose $x_1 \leftarrow 1$, $x_4 \leftarrow 0$
- solvable with *unit propagation*

Horn SAT

Horn formulas

implication $(z \wedge w \wedge q) \Rightarrow u$. LHS is all positive, RHS one positive literal

negative clauses $(\bar{x} \vee \bar{w} \vee \bar{y})$. All literals negated.

Horn formulas as CNF

- negative clauses are already CNF

Horn formulas as CNF

- negative clauses are already CNF
- implications use the following transformations

$$(\bigwedge_{i=1}^{k} x_i) \Rightarrow y$$

$$\neg(\bigwedge_{i=1}^{k} x_i) \vee y$$

$$(\bigvee_{i=1}^{k} \bar{x}_i) \vee y$$

Horn formulas as CNF

- negative clauses are already CNF
- implications use the following transformations

$$\left(\bigwedge_{i=1}^{k} x_i\right) \Rightarrow y$$
$$\neg\left(\bigwedge_{i=1}^{k} x_i\right) \vee y$$
$$\left(\bigvee_{i=1}^{k} \bar{x}_i\right) \vee y$$

- So we can think about special CNF with at most one positive literal.

Unit propagation

function $\textsc{UnitProp}(S$: Set of clauses)
  while S has a unit clause $C = \{ z \}$ **do**
    if $z = \bar{x}_i$ **then**
      $x_i \leftarrow 0$
    else
      $x_i \leftarrow 1$
    end if
    $S \leftarrow \{ C \mid C \in S, z \notin C \}$
    If $S = \emptyset$, return SATISFIABLE
    $S \leftarrow \{ C \setminus \{ \neg z \} \mid C \in S \}$
    If $\emptyset \in S$, return UNSATISFIABLE
  end while
  return S

Unit propagation

function $\text{UNITPROP}(S$: Set of clauses)

 while S has a unit clause $C = \{ z \}$ **do**

  if $z = \bar{x}_i$ **then**

   $x_i \leftarrow 0$

  else

   $x_i \leftarrow 1$

  end if

  $S \leftarrow \{ C \mid C \in S, z \notin C \}$

  If $S = \emptyset$, return SATISFIABLE

  $S \leftarrow \{ C \setminus \{ \neg z \} \mid C \in S \}$

  If $\emptyset \in S$, return UNSATISFIABLE

 end while

 return S

Solving Horn SAT with Unit Propagation

Procedure HornProp

1. Apply unit propagation
2. If no contradiction is detected, set the remaining variables to false.

Claim 1

If the procedure detects a contradiction, the instance is unsatisfiable

Solving Horn SAT with Unit Propagation

Procedure HornProp

1. Apply unit propagation
2. If no contradiction is detected, set the remaining variables to false.

Claim 1

If the procedure detects a contradiction, the instance is unsatisfiable

Claim 2

If no contradiction is detected, the resulting assignment is valid

Solving Horn SAT with Unit Propagation

Claim 1

If the procedure detects a contradiction, the instance is unsatisfiable

Claim 2

If no contradiction is detected, the resulting assignment is valid

Proof

any remaining clause has at least one negative literal

Another NP-Complete problem!

The longest path

The world's longest path (Sendero de Chile): 9,700 km.
(originally scheduled for completion in 2010; now delayed until 2038)

That's all, folks: keep searching!

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!

If you said P is NP tonight,
There would still be papers left to write.
I have a weakness;
I'm addicted to completeness,
And I keep searching for the longest path.

The algorithm I would like to see
Is of polynomial degree.
But it's elusive:
Nobody has found conclusive
Evidence that we can find a longest path.

I have been hard working for so long.
I swear it's right, and he marks it wrong.
Some how I'll feel sorry when it's done: GPA 2.1
Is more than I hope for.

Garey, Johnson, Karp and other men (and women)
Tried to make it order N log N.
Am I a mad fool
If I spend my life in grad school,
Forever following the longest path?

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path.

Written by Dan Barrett in 1988 while a student
at Johns Hopkins during a difficult algorithms take-home final

https://www.cs.princeton.edu/courses/archive/fall12/cos226/lectures