GRASP-an efficient SAT solver

Pankaj Chauhan

What is SAT?

- Given a propositional formula in CNF, find an assignment to boolean variables that makes the formula true!
- E.g.

$w_1 = (x_2\ Ú\ x_3)$

$w_2 = (Øx_1\ Ú\ Øx_4)$

$w_3 = (Øx_2\ Ú\ x_4)$

$A = \{x_1=0, x_2=1, x_3=0, x_4=1\}$

SATisfying assignment!

What is SAT?

- Solution 1: Search through all assignments!
  - n variables $\rightarrow 2^n$ possible assignments, explosion!
- SAT is a classic NP-Complete problem, solve SAT and P=NP!

Why SAT?

- Fundamental problem from theoretical point of view
- Numerous applications
  - CAD, VLSI
  - Optimization
  - Model Checking and other type of formal verification
  - AI, planning, automated deduction

Outline

- Terminology
- Basic Backtracking Search
- GRASP
- Pointers to future work

Please interrupt me if anything is not clear!

Terminology

- CNF formula *j*
  - $x_1,\ldots, x_n$: n variables
  - $w_1,\ldots, w_m$: m clauses

w_1 = (x_2 Ú x_3)

w_2 = (Øx_1 Ú Øx_4)

w_3 = (Øx_2 Ú x_4)

$A = \{x_1=0, x_2=1, x_3=0, x_4=1\}$

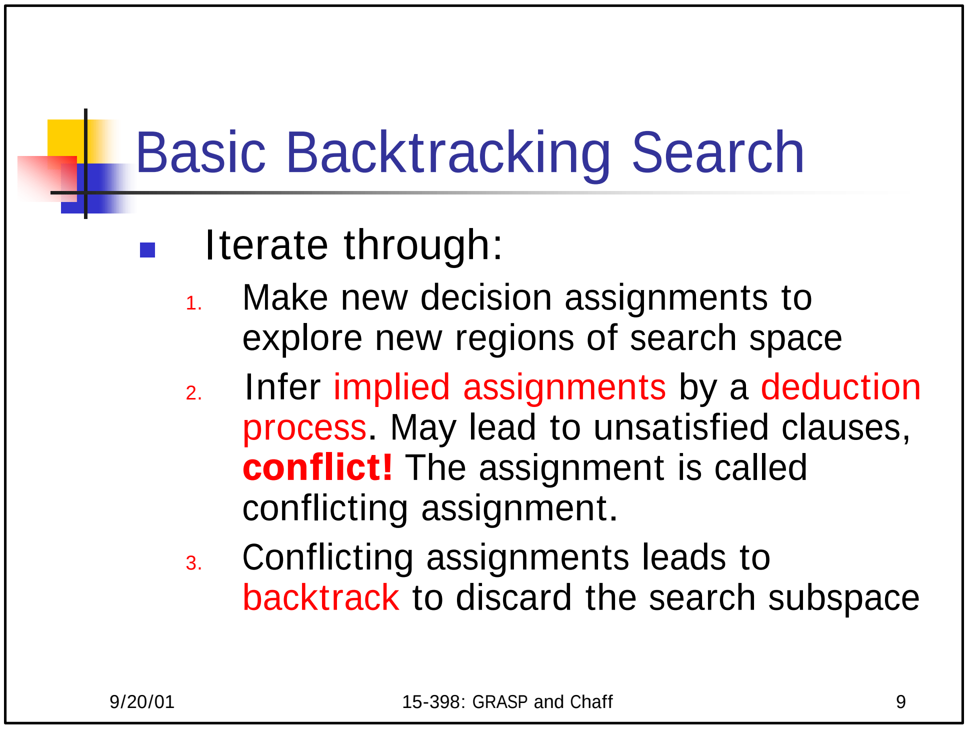
- Assignment *A*
  - Set of $(x,v(x))$ pairs
  - $|A| < n$ ® partial assignment $\{(x_1,0), (x_2,1), (x_4,1)\}$
  - $|A| = n$ ® complete assignment $\{(x_1,0), (x_2,1), (x_3,0), (x_4,1)\}$
  - $j\,|_A= 0$ ® unsatisfying assignment $\{(x_1,1), (x_4,1)\}$
  - $j\,|_A= 1$ ® satisfying assignment $\{(x_1,0), (x_2,1), (x_4,1)\}$

Terminology

- Assignment A *(contd.)*
  - $j\,|_A = X$ ® unresolved $\{(x_1,0), (x_2,0), (x_4,1)\}$
- An assignment partitions the clause database into three classes
  - Satisfied, unsatisfied, unresolved
- Free literals: unassigned literals of a clause
- Unit clause: #free literals = 1

Basic Backtracking Search

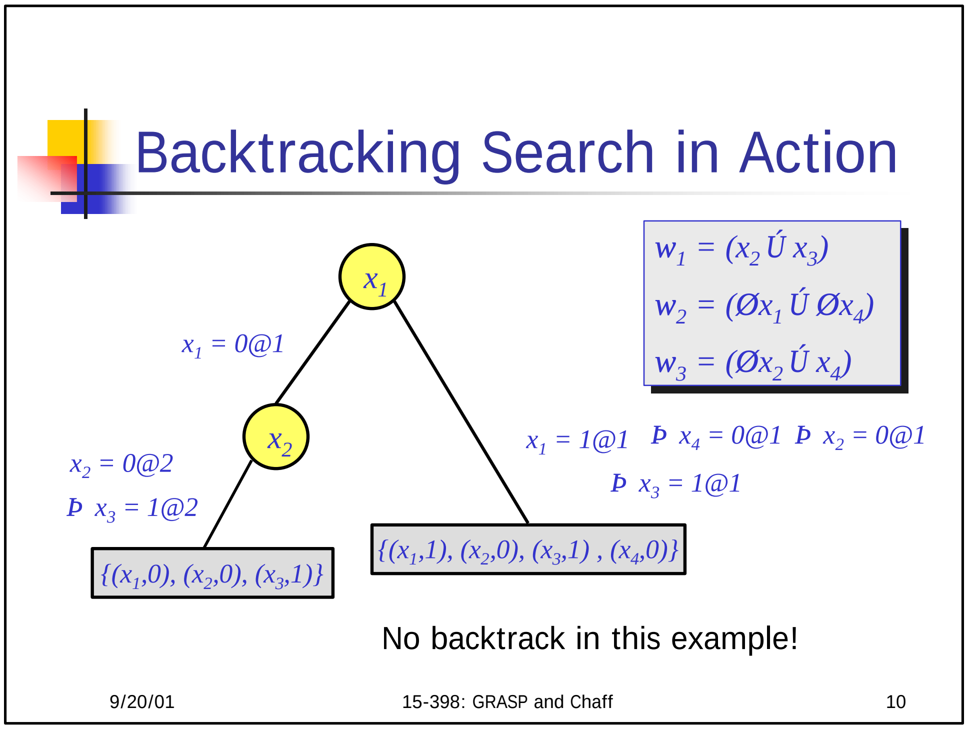
- Organize the search in the form of a decision tree
  - Each node is an assignment, called decision assignment
  - Depth of the node in the decision tree $\rightarrow$ decision level $d(x)$
  - $x=v@d \rightarrow x$ is assigned to v at decision level d



Basic Backtracking Search

- Iterate through:
  1. Make new decision assignments to explore new regions of search space
  2. Infer implied assignments by a deduction process. May lead to unsatisfied clauses, **conflict!** The assignment is called conflicting assignment.
  3. Conflicting assignments leads to backtrack to discard the search subspace

9/20/01 15-398: GRASP and Chaff 9

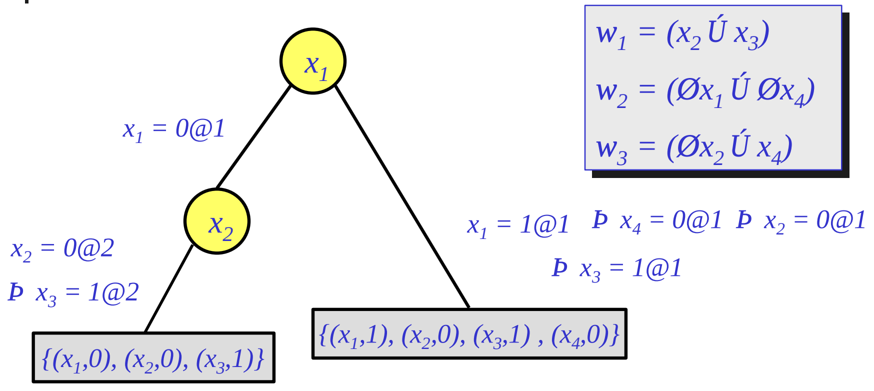


Backtracking Search in Action

$w_1 = (x_2 \vee x_3)$

$w_2 = (\neg x_1 \vee \neg x_4)$

$w_3 = (\neg x_2 \vee x_4)$



x_1
$x_1 = 0@1$
x_2
$x_2 = 0@2$
$\Rightarrow x_3 = 1@2$
$\{(x_1,0), (x_2,0), (x_3,1)\}$
$x_1 = 1@1 \;\; \Rightarrow x_4 = 0@1 \;\; \Rightarrow x_2 = 0@1$
$\Rightarrow x_3 = 1@1$
$\{(x_1,1), (x_2,0), (x_3,1), (x_4,0)\}$

No backtrack in this example!

9/20/01 15-398: GRASP and Chaff 10

Backtracking Search in Action

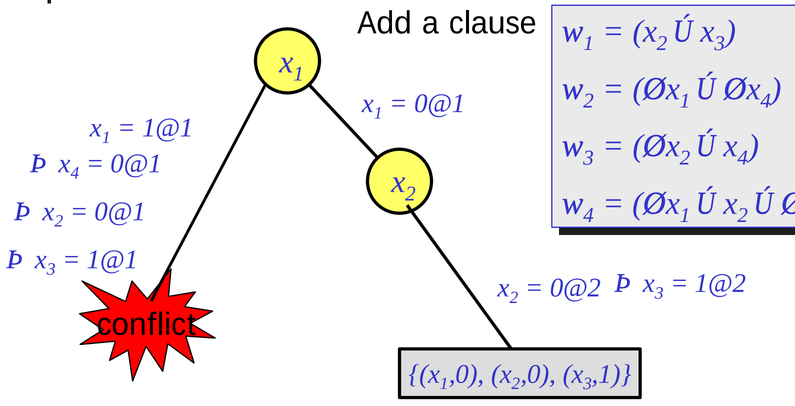
Add a clause

w_1 = (x_2 Ú x_3)

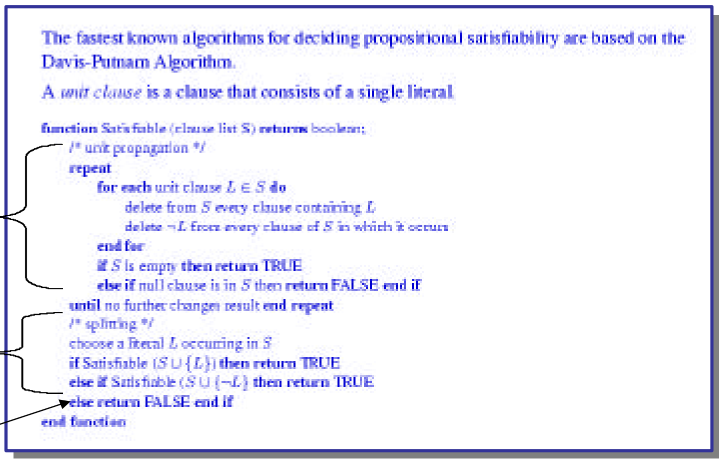
w_2 = (Øx_1 Ú Øx_4)

w_3 = (Øx_2 Ú x_4)

w_4 = (Øx_1 Ú x_2 Ú Øx_3)



Davis-Putnam revisited



The fastest known algorithms for deciding propositional satisfiability are based on the Davis-Putnam Algorithm.

A *unit clause* is a clause that consists of a single literal.

```
function Satisfiable (clause list S) returns boolean;
    /* unit propagation */
    repeat
        for each unit clause $L \in S$ do
            delete from $S$ every clause containing $L$
            delete $\neg L$ from every clause of $S$ in which it occurs
        end for
        if $S$ is empty then return TRUE
        else if null clause is in $S$ then return FALSE end if
    until no further changes result end repeat
    /* splitting */
    choose a literal $L$ occurring in $S$
    if Satisfiable ($S \cup \{L\}$) then return TRUE
    else if Satisfiable ($S \cup \{\neg L\}$) then return TRUE
    else return FALSE end if
end function
```

Deduction

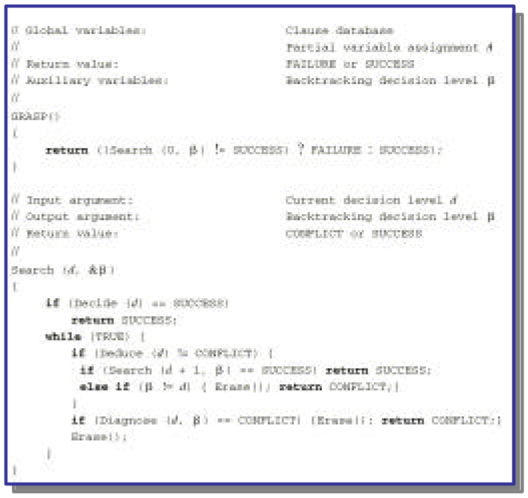
Decision

Backtrack

GRASP

- GRASP is Generalized seaRch Algorithm for the Satisfiability Problem (Silva, Sakallah, ’96)
- Features:
  - Implication graphs for BCP and conflict analysis
  - Learning of new clauses
  - **Non-chronological** backtracking!

GRASP search template



```
// Global variables:                    Clause database
//                                      Partial variable assignment A
// Return value:                        FAILURE or SUCCESS
// Auxiliary variables:                 Backtracking decision level β
//
GRASP()
{
    return ((Search (0, β) != SUCCESS) ? FAILURE : SUCCESS);
}

// Input argument:                      Current decision level d
// Output argument:                     Backtracking decision level β
// Return value:                        CONFLICT or SUCCESS
//
Search (d, &β )
{
    if (Decide (d) == SUCCESS)
        return SUCCESS;
    while (TRUE) {
        if (Deduce (d) != CONFLICT) {
         if (Search (d + 1, β) == SUCCESS) return SUCCESS;
         else if (β != d) { Erase(); return CONFLICT;}
        }
        if (Diagnose (d, β ) == CONFLICT) {Erase(); return CONFLICT;}
        Erase();
    }
}
```

GRASP Decision Heuristics

- Procedure `decide()`
- Choose the variable that satisfies the most #clauses == max occurences as unit clauses at current decision level
- Other possibilities exist

GRASP Deduction

- Boolean Constraint Propagation using implication graphs

  E.g. for the clause *w = (x Ú Øy)*, if *y=1*, then we must have *x=1*
- For a variable *x* occuring in a clause , assignment 0 to all other literals is called antecedent assignment *A(x)*
  - E.g. for *w = (x Ú y Ú Øz)*,

    A(x) = {(y,0), (z,1)}, A(y) = {(x,0),(z,1)}, A(z) = {(x,0), (y,0)}
  - Variables directly responsible for forcing the value of *x*
  - Antecedent assignment of a decision variable is empty

Implication Graphs

- Nodes are variable assignments $x=v(x)$ (decision or implied)
- Predecessors of x are antecedent assignments $A(x)$
  - No predecessors for decision assignments!
- Special conflict vertices have $A(k)$ = assignments to vars in the unsatisfied clause
- Decision level for an implied assignment is

$$\boxed{d(x) = max\{d(y)|(y,v(y))\ Î\ A(x)\}}$$

Example Implication Graph

Current truth assignment: $\{x_9=0@1\ ,x_{10}=0@3, x_{11}=0@3, x_{12}=1@2, x_{13}=1@2\}$

Current decision assignment: $\{x_1=1@6\}$

$w_1 = (Øx_1\ Ú\ x_2)$

$w_2 = (Øx_1\ Ú\ x_3\ Ú\ x_9)$

$w_3 = (Øx_2\ Ú\ Øx_3\ Ú\ x_4)$

$w_4 = (Øx_4\ Ú\ x_5\ Ú\ x_{10})$

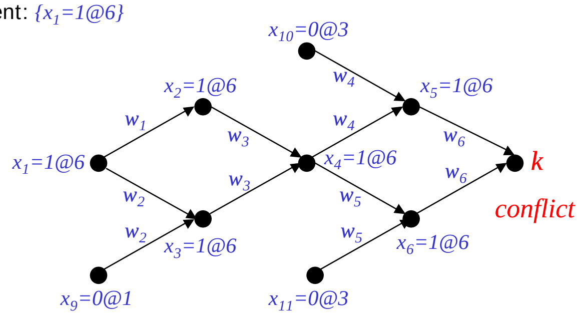
$w_5 = (Øx_4\ Ú\ x_6\ Ú\ x_{11})$

$w_6 = (Øx_5\ Ú\ x_6)$

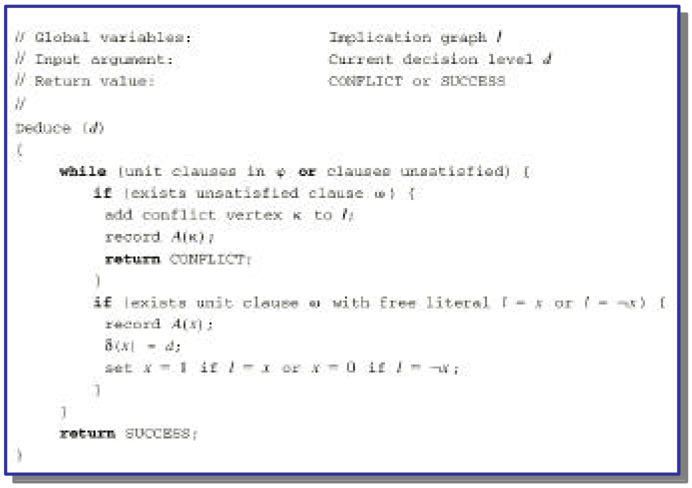
$w_7 = (x_1\ Ú\ x_7\ Ú\ Øx_{12})$

$w_8 = (x_1 Ú\ x_8)$

$w_9 = (Øx_7\ Ú\ Øx_8\ Ú\ Ø\ x_{13})$



GRASP Deduction Process



```
// Global variables:              Implication graph I
// Input argument:                Current decision level d
// Return value:                  CONFLICT or SUCCESS
//
Deduce (d)
{
    while (unit clauses in φ or clauses unsatisfied) {
        if (exists unsatisfied clause ω) {
          add conflict vertex κ to I;
          record A(κ);
          return CONFLICT;
        }
        if (exists unit clause ω with free literal l = x or l = ¬x) {
          record A(x);
          δ(x) = d;
          set x = 1 if l = x or x = 0 if l = ¬x;
        }
    }
    return SUCCESS;
}
```

GRASP Conflict Analysis

- After a conflict arises, analyze the implication graph at current decision level
- Add new clauses that would prevent the occurrence of the same conflict in the future $\Rightarrow$ Learning
- Determine decision level to backtrack to, might not be the immediate one $\Rightarrow$ Non-chronological backtrack

Learning

- Determine the assignment that caused the conflict, negation of this assignment is called conflict induced clause $w_C(k)$
  - The conjunct of this assignment is necessary condition for k
  - So adding $w_C(k)$ will prevent the occurrence of k again

Learning

- Find $w_C(k)$ by a backward traversal of the IG, find the roots of the IG in the transitive fanin of k
- For our example IG,

$$w_C(k) = (\neg x_1 \vee x_9 \vee x_{10} \vee x_{11})$$

Learning (some math)

- For any node of an IG x, partition $A(x)$ into

$$\boxed{\begin{array}{l} L(x) = \{(y,v(y)) \in A(x) \mid d(y)<d(x)\} \\ S(x) = \{(y,v(y)) \in A(x) \mid d(y)=d(x)\} \end{array}}$$

- Conflicting assignment $A_C(k) = causesof(k)$, where

$$\boxed{causesof(x) = \begin{cases} (x,v(x)) & \text{if } A(x) = \phi \\ L(x) \cup \left[\bigcup_{(y,v(y)) \in S(x)} causesof(y) \right] & \text{o/w} \end{cases}}$$

Learning (some math)

- Deriving conflicting clause $w_C(k)$ from the conflicting assignment $A_C(k)$ is straight forward

$$\boxed{w_C(k) = \Sigma_{(x,v(x)) \in A_C(k)} x^{v(x)}}$$

- For our IG,

$$A_C(k) = \{x_1 = 1@6,\ x_9 = 0@1,\ x_{10} = 0@3,\ x_{11} = 0@3\}$$

Learning

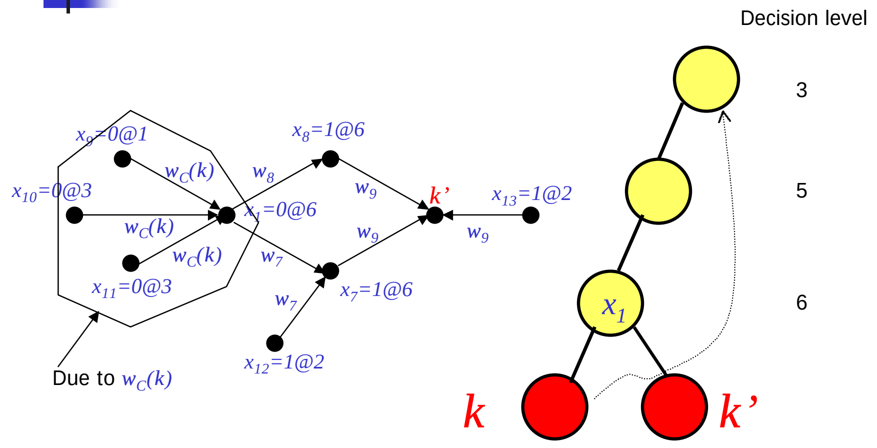
- Unique implication points (UIPs) of an IG also provide conflict clauses
- Learning of new clauses increases clause database size
- Heuristically delete clauses based on a user parameter
    - If size of learned clause > parameter, don’t include it

Backtracking

Failure driven assertions (FDA):

- If $w_C(k)$ involves current decision variable, then after addition, it becomes unit clause, so different assignment for the current variable is immediately tried.
- In our IG, after erasing the assignment at level 6, $w_C(k)$ becomes a unit clause Øx_1
- This immediately implies $x_1=0$

Continued IG



Backtracking

Conflict Directed Backtracking

- Now deriving a new IG after setting $x_1=0$ by FDA, we get another conflict k'
- Non-chronological backtrack to decision level 3, because backtracking to any level 5, 4 would generate the same conflict k'

Backtracking

Conflict Directed Backtracking contd.

$A_C(k') = \{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2\}$

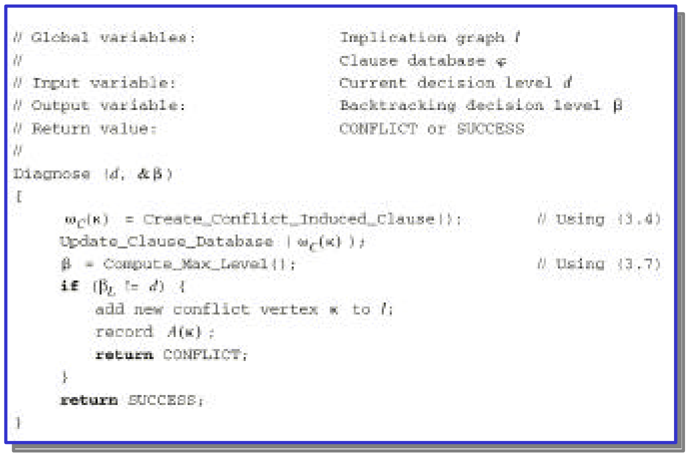
$w_C(k) = (x_9$ Ú x_{10} Ú x_{11} Ú Ø x_{12} Ú Ø $x_{13})$

- Backtrack level is given by

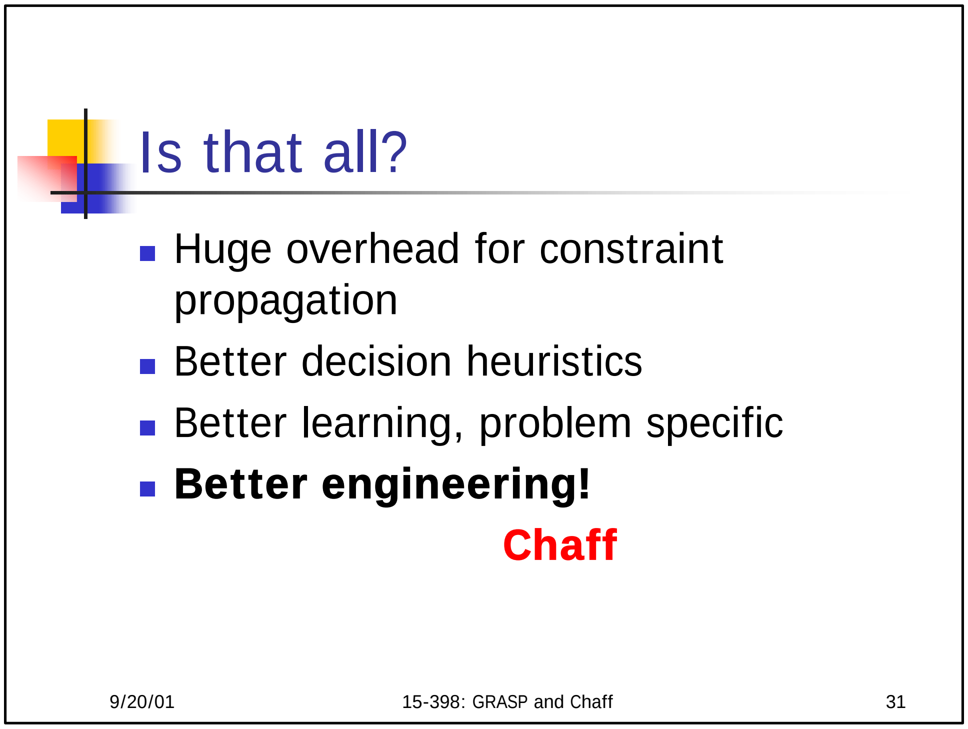
$b = max\{d(x)|(x,v(x))$ Î $A_C(k')\}$

- $b = d\text{-}1$ chronological backtrack
- $b < d\text{-}1$ non-chronological backtrack

Procedure Diagnose()



```
// Global variables:                 Implication graph I
//                                   Clause database φ
// Input variable:                   Current decision level d
// Output variable:                  Backtracking decision level β
// Return value:                     CONFLICT or SUCCESS
//
Diagnose (d, &β)
{
    ω_C(κ) = Create_Conflict_Induced_Clause();        // Using (3.4)
    Update_Clause_Database (ω_C(κ));
    β = Compute_Max_Level();                          // Using (3.7)
    if (β_L != d) {
        add new conflict vertex κ to I;
        record A(κ);
        return CONFLICT;
    }
    return SUCCESS;
}
```



Is that all?

- Huge overhead for constraint propagation
- Better decision heuristics
- Better learning, problem specific
- **Better engineering!**

Chaff