Artificial Intelligence

Constraint Satisfaction Problems CSP

More about

Textbook, Chapter 5, Constraint Satisfaction Problems

Outline

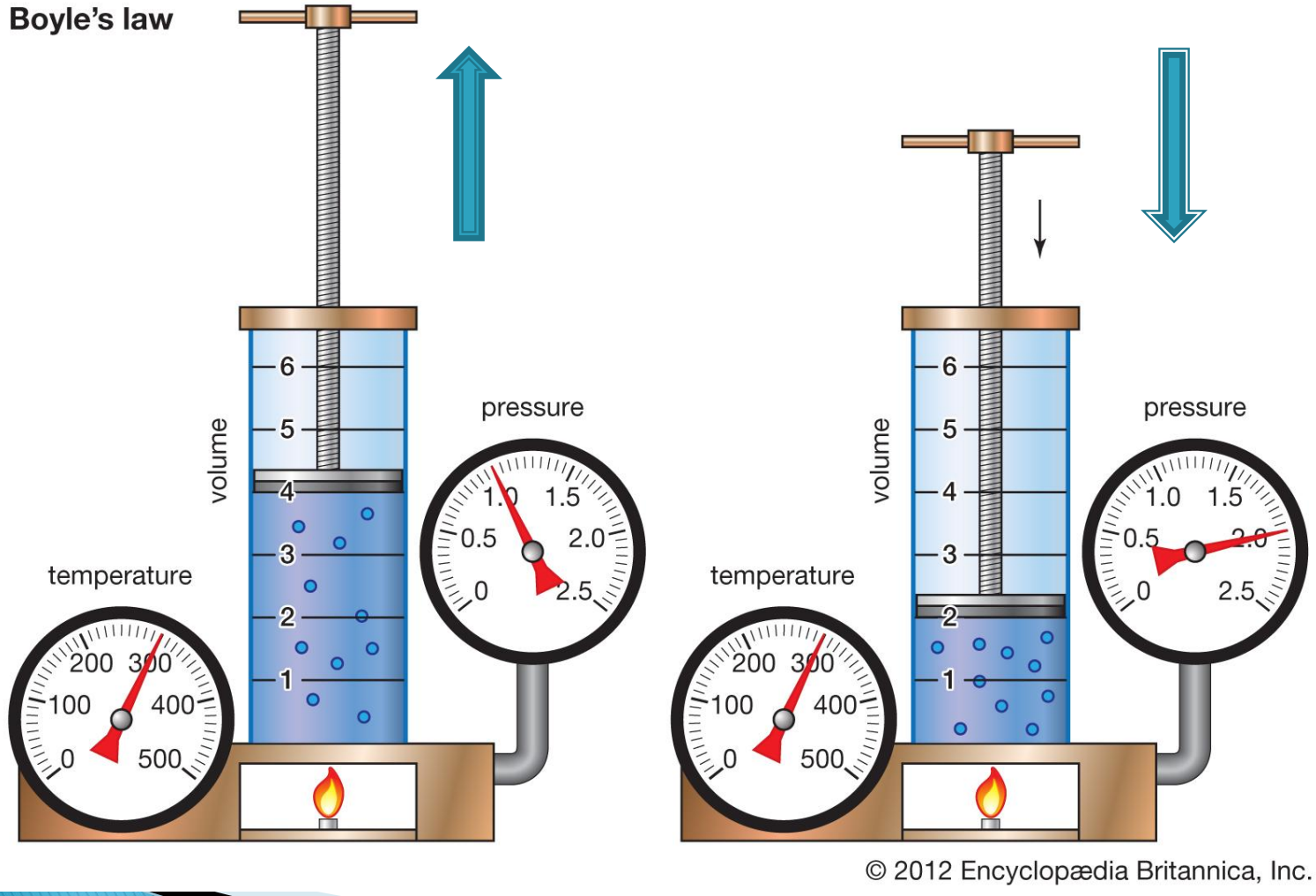
- What is CSP?
- Toy CSP and real life CSP
- CSP as search
  - Backtracking
  - Heuristics
    - Variable ordering and value selection
    - Forward checking
    - Constraint propagation
- Improved CSP representations
- Complexity of solving tree structured CSPs

Constraints

- Physical laws
- Resources
- Schedules
- User demands
- Economics
- Safety regulations, standards, codes, etc.
- …
- Logic constraints
- Arithmetic constraints
- Geometrical constraints
- …

Constraints

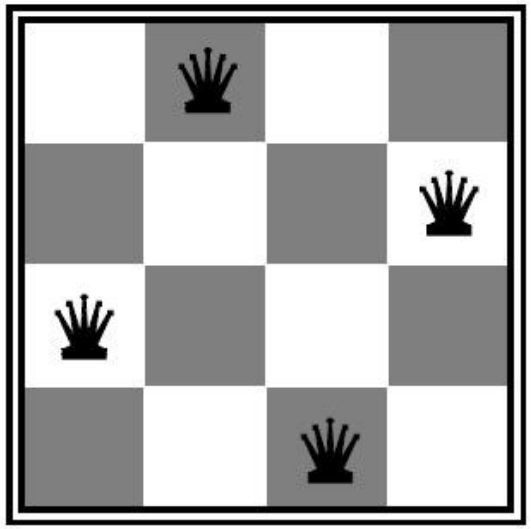
$$PV = nRT$$

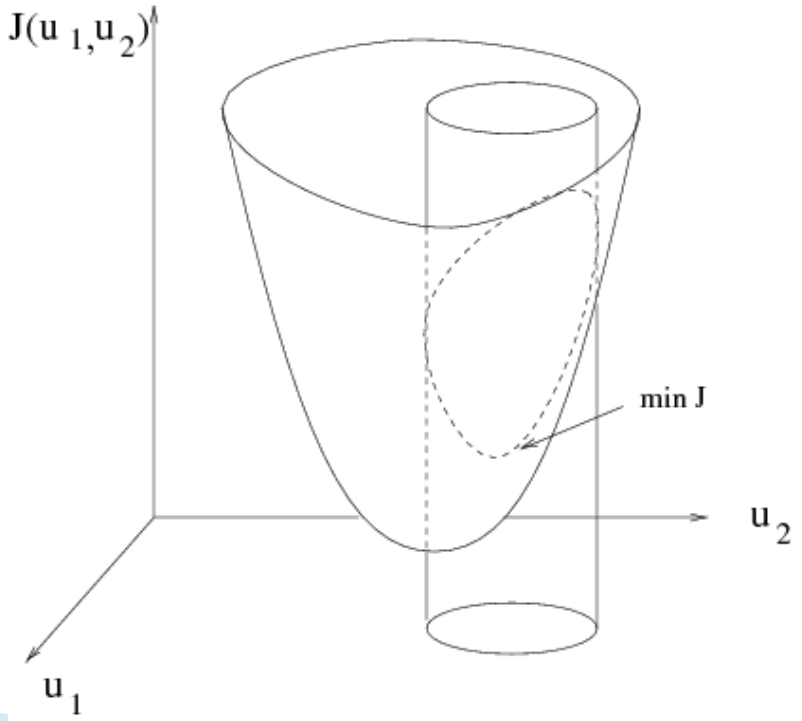


Constraints

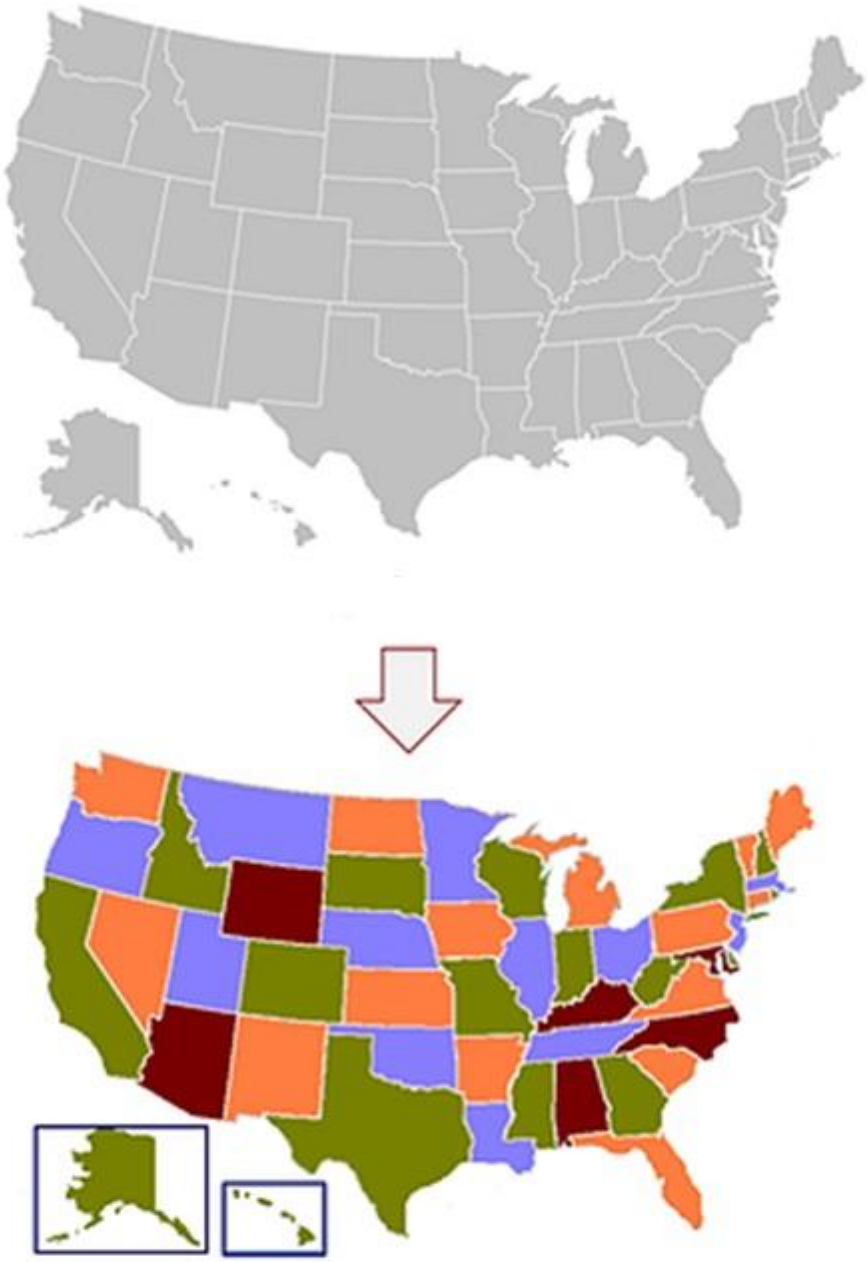
	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2





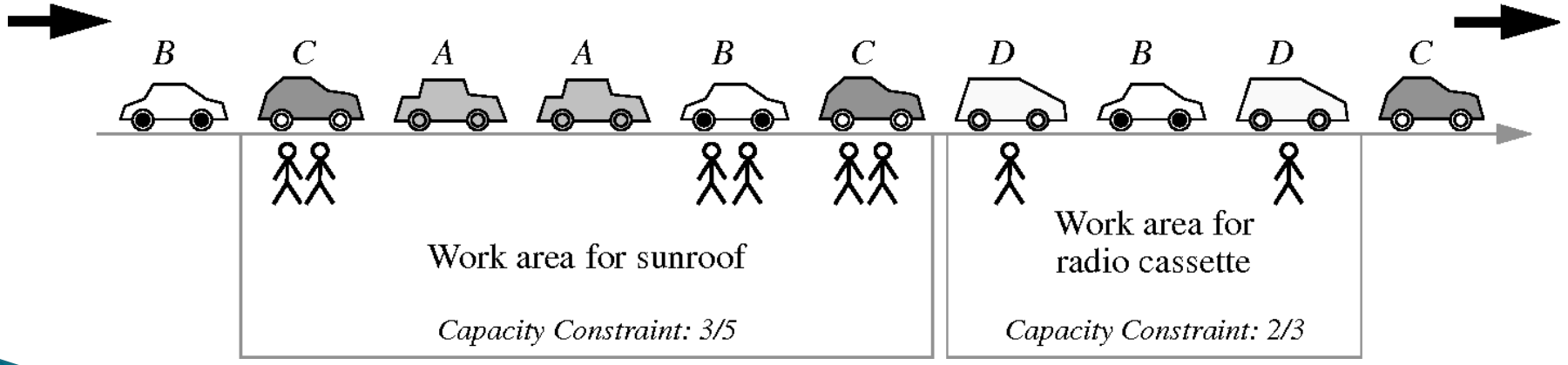
Constraints



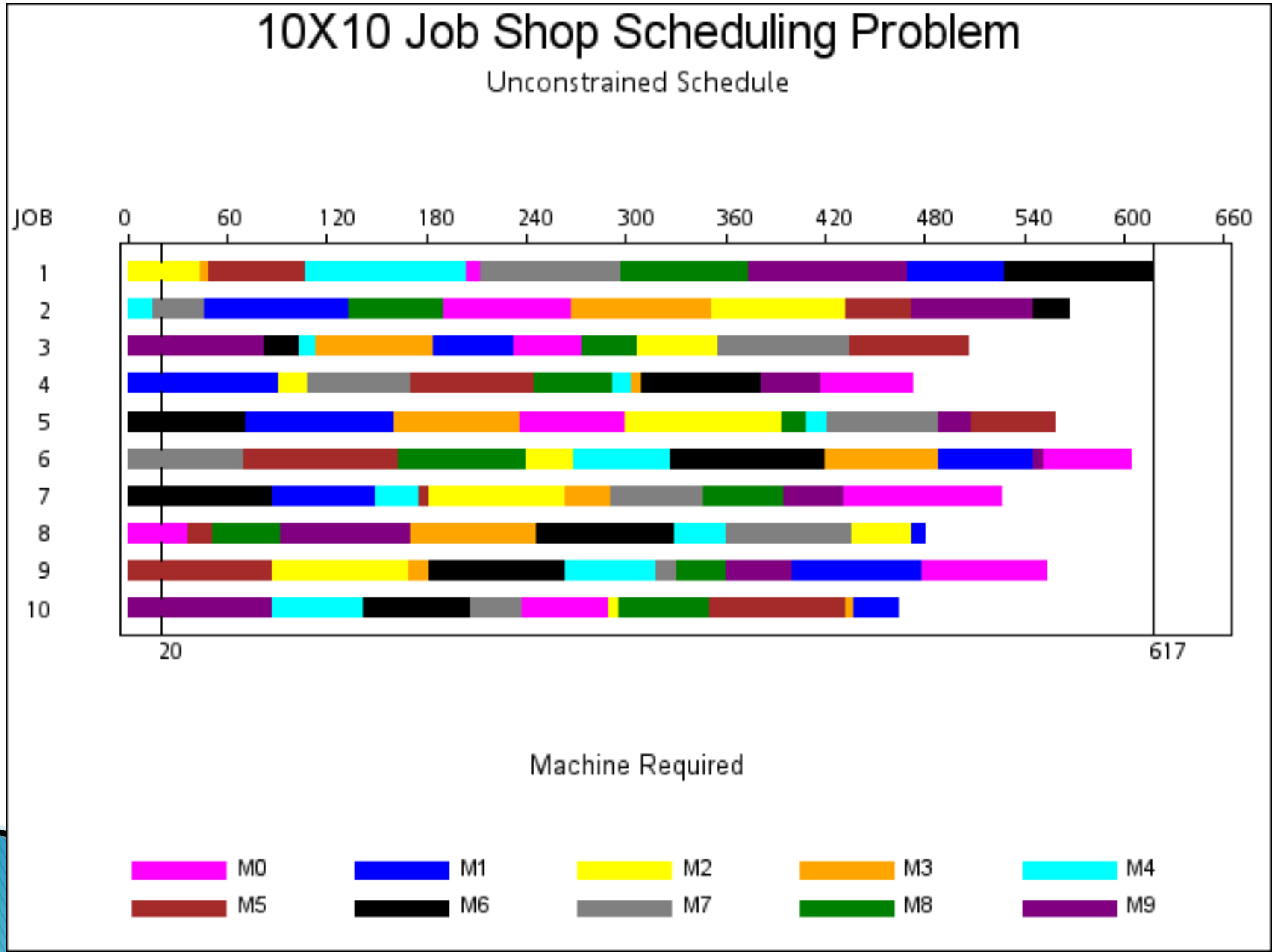
Constraints

Production Requirements:

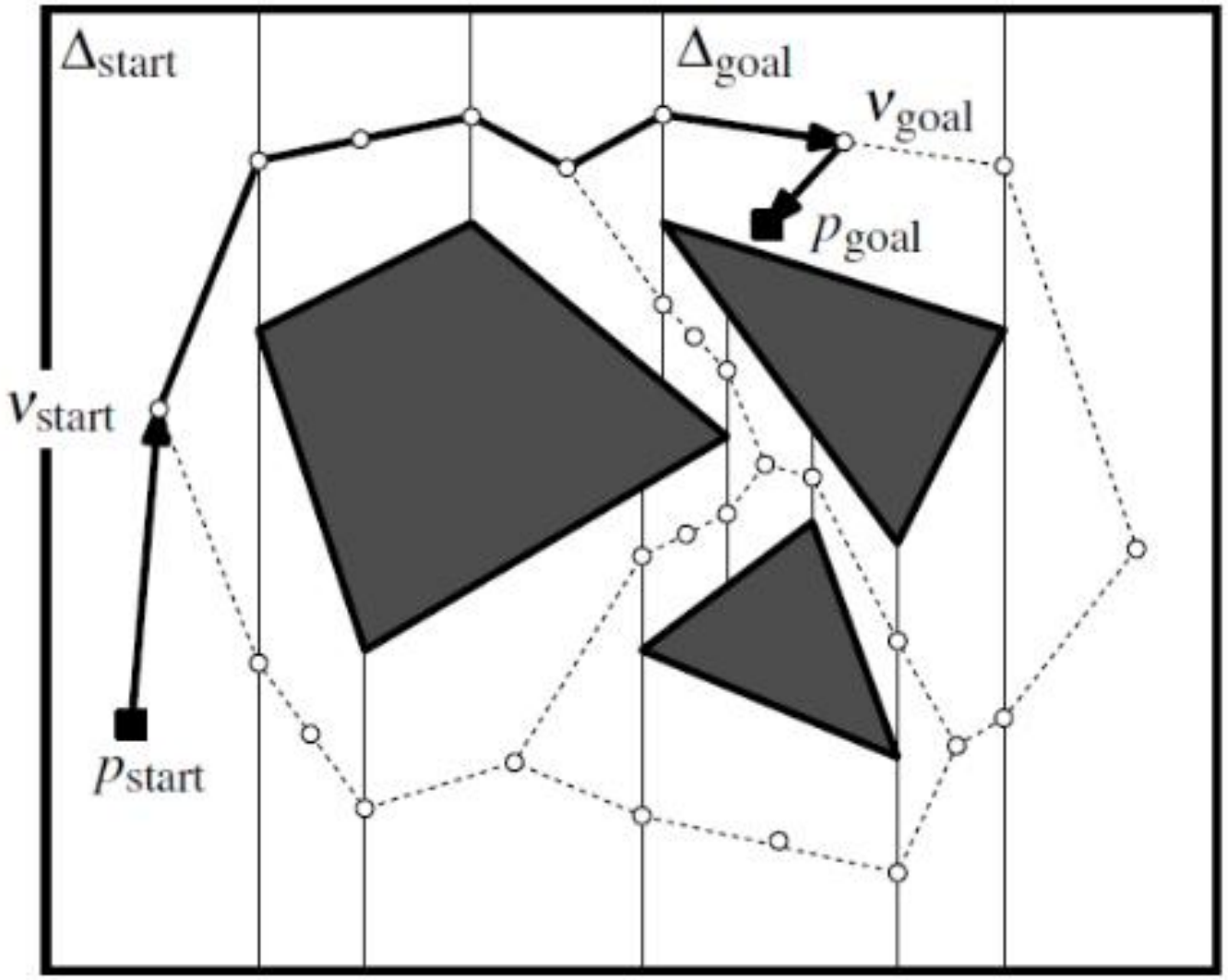
Options (✓ = required, × = not):	Model A	Model B	Model C	Model D	
					
Sunroof	×	✓	✓	×	
Radio cassette	✓	×	✓	✓	
Air-conditioning	✓	✓	×	✓	
Anti-rust treatment	×	✓	✓	✓	
Power brakes	✓	×	✓	×	
Number of cars required:	30	30	20	40	Total: 120



Constraints



Constraints



Constraints

Hardware/ software formal verification, protocol design, etc.

Satisfability of logical expressions (SAT)
- what is the set of values of (binary) variables, which makes all stated logical expressions true in the same time?

$$(a \vee \neg b \vee \neg c) \text{ AND } (b \vee \neg c) \text{ AND } (a \vee c)$$

a = True/ False?
b = True/ False?
c = True/ False?

$2 \times 2 \times 2 = 2^3 = 8$

here: a = True, b = True, c don't care, or
a = True, c = False, b don't care,
etc.

Variables: 3

Clauses: 3

Constraints

IBM Load-Store Unit (LSU)

The instance bmc-ibm-6.cnf, IBM LSU 1997:

```
p cnf 51639 368352
-1 7 0
-1 6 0
-1 5 0
-1 -4 0
-1 3 0
-1 2 0
-1 -8 0
-9 15 0
-9 14 0
-9 13 0
-9 -12 0
-9 11 0
-9 10 0
-9 -16 0
-17 23 0
-17 22 0
```

i.e., ((not x_1) or x_7) AND

((not x_1) or x_6) AND

etc.

Should x_1 be set to False??

BMC –
Bounded Model Checking

Variables: 51654

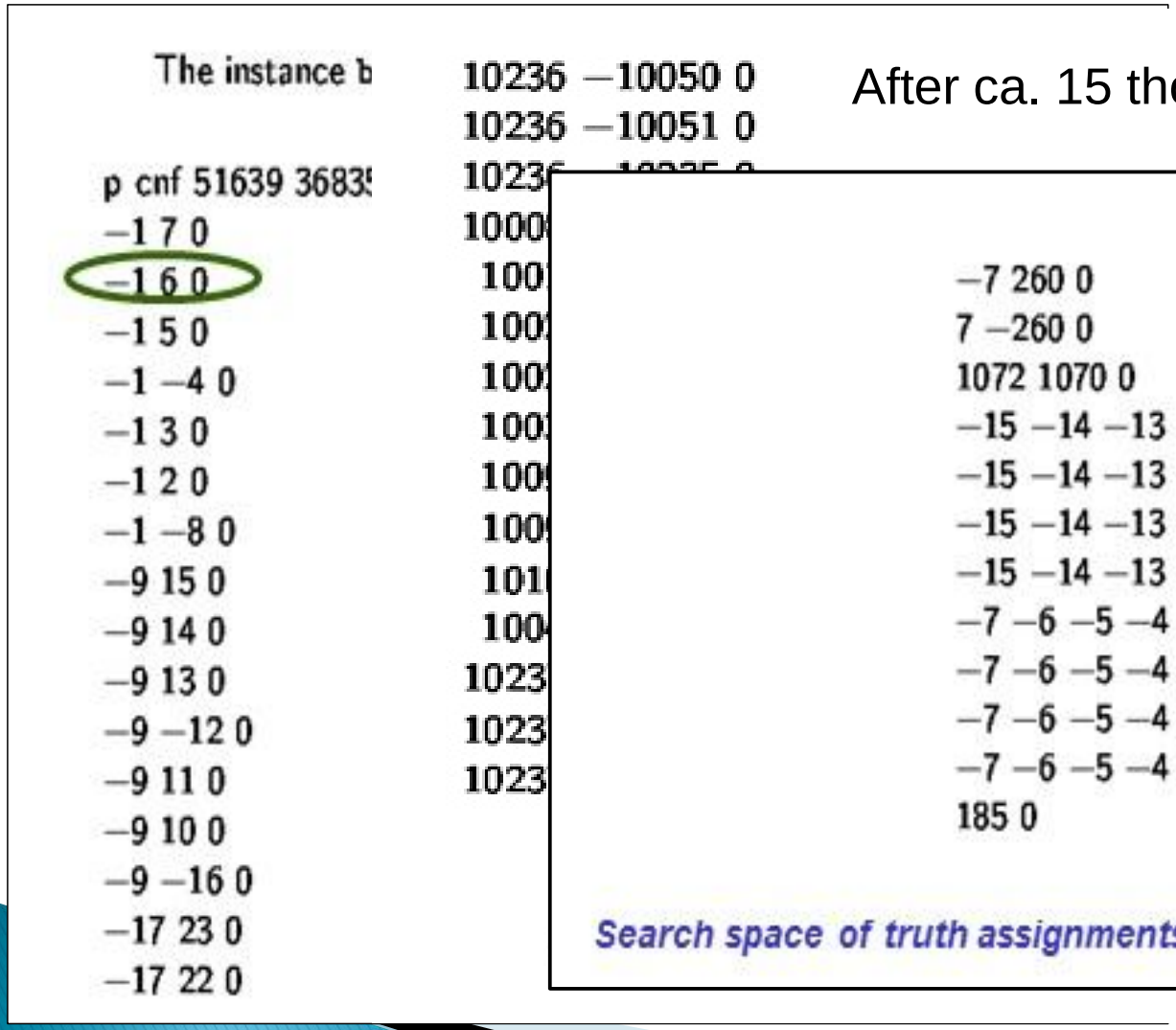
Clauses: 368367

Temporal bound: 22

https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/BMC/description.html

Constraints

Hardware/ software formal verification, protocol design, etc.



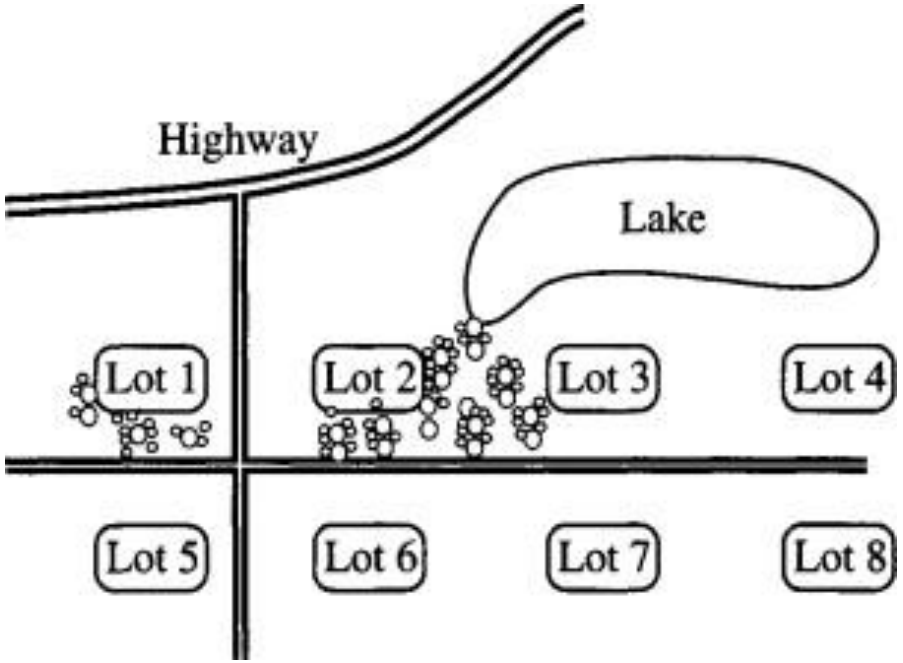
After ca. 15 thousand pages later

Solved in ca. 10 msec!

Constraints

Allocation problem

Five developments are to be located on the lots:
(1) a recreation area,
(2) an apartment complex,
(3) a cluster of 50 single-family houses,
(4) a large cemetery, and
(5) a dump site.



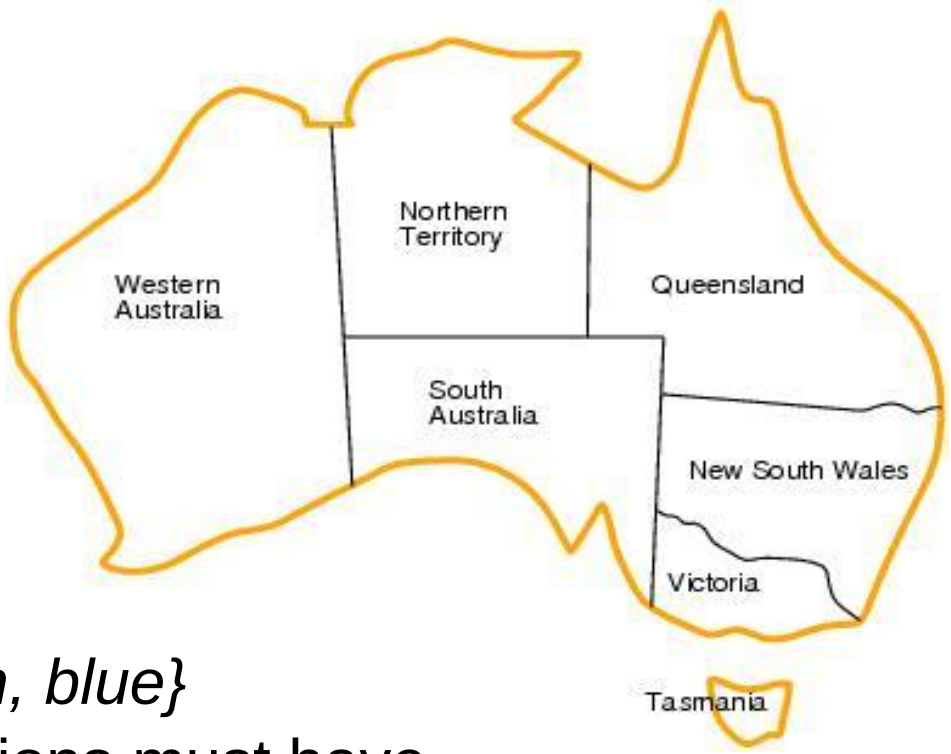
Conditions:

-The recreation area must be near the lake.
- Steep slopes must be avoided for all but the recreation area.
- Poor soil must be avoided for developments with construction (apartments, houses).
- The highway must not be near the apartments, the houses, or the recreation area.
- The dump site must not be visible from the apartments, the houses, or the lake.
- Lots 3 and 4 have poor soil.
- Lots 3, 4, 7, and 8 are on steep slopes.
- Lots 2, 3, and 4 are near the lake.
- Lots 1 and 2 are near the highway.
- No two developments may occur on the same lot.

Constraint satisfaction problem

- What is a CSP?
  - Finite set of variables $V_1, V_2, \ldots, V_n$
  - Finite set of constraints $C_1, C_2, \ldots, C_m$
  - Nonempty domain of possible values for each variable $D_{V1}, D_{V2}, \ldots D_{Vn}$
  - Each constraint C_i limits the values that variables can take, e.g., $V_1 \neq V_2$
- A *state* is defined as an *assignment* of values to some or to all variables.
- *Consistent assignment*: does not violate the constraints.
- An assignment is *complete* when every variable is set.
- A *solution* to a CSP is a complete assignment that satisfies all constraints.
- Some CSPs require a solution that maximizes an *objective function*.

CSP example: map coloring

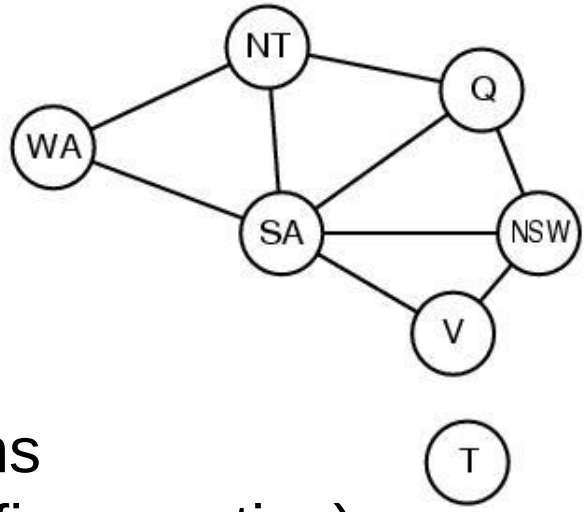


- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: $D_i = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors.
  - E.g. $WA \neq NT$, $WA \neq SA$, $SA \neq NT$, …
  - E.g. *(WA,NT) ≠ (red, green), (red, blue), …}*
  - *etc.*

CSP example: map coloring

- Solutions are assignments satisfying all constraints, e.g.
  {WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green}

Constraint graph



- Benefits of CSP formulation
  - Standard representation pattern
  - Generic goal and successor functions
  - Generic heuristics (no domain specific expertise).
- Constraint graph = nodes are variables, edges show constraints.
  - Graph can be used to simplify search.
    - e.g. Tasmania is an independent subproblem.

Varieties of CSPs

- Discrete variables
  - Finite domains; size $d \Rightarrow O(d^n)$ complete assignments.
    - e.g. Boolean CSP, includes Boolean satisfiability (NP-complete).
  - Infinite domains (integers, strings, etc.)
    - e.g. job scheduling, variables are start/end days for each job
    - Need a constraint language

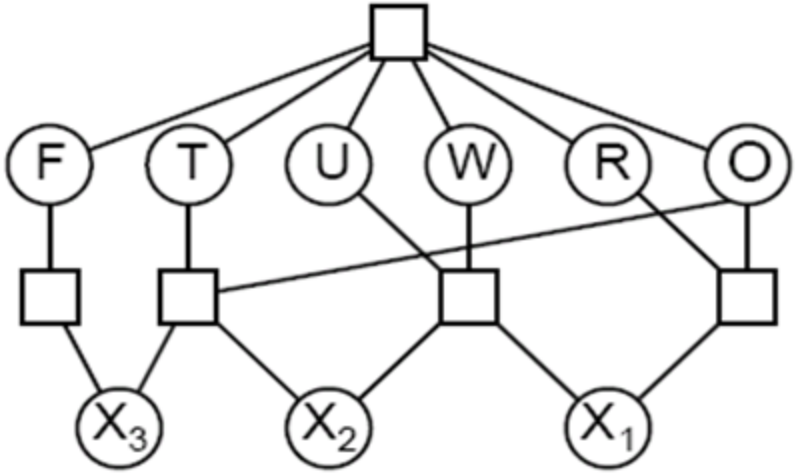
      e.g $StartJob_1 + 5 \leq StartJob_3$.
    - Linear constraints solvable, nonlinear undecidable.
- Continuous variables
  - e.g. start/end times for Hubble Telescope observations.
  - Linear constraints solvable in poly time by LP methods.

Varieties of constraints

- Unary constraints involve a single variable.
  - e.g. $SA \neq green$
- Binary constraints involve pairs of variables.
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables.
  - e.g. cryptharithmetic column constraints.
- Preference (soft constraints) e.g. *red* is better than *green* often representable by a cost for each variable assignment $\rightarrow$ constrained optimization problems.

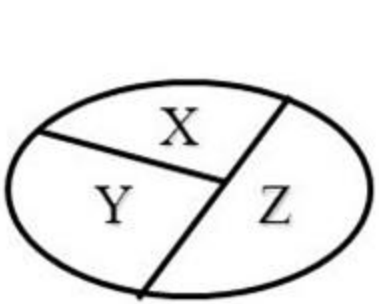
Example; cryptharithmetic

```
  T W O
+ T W O
-------
F O U R
```

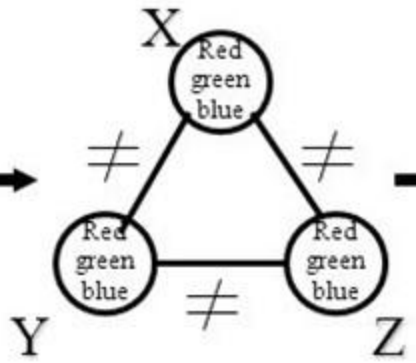


- Variables: *F, T, U, W, R, O,* X_1, X_2, X_3
- Domains: $D_i = \{0, 1, 2, 3, 4, 5, 6 , 7, 8, 9\}$
  - $D_j = \{0, 1\}$
- Constraints:
  alldiff(*F*,*T*,*U*,*W*,*R*,*O*)
  $O + O = R + 10 \times X_1$, etc.

Constraint satisfaction problems



Coloring Problem



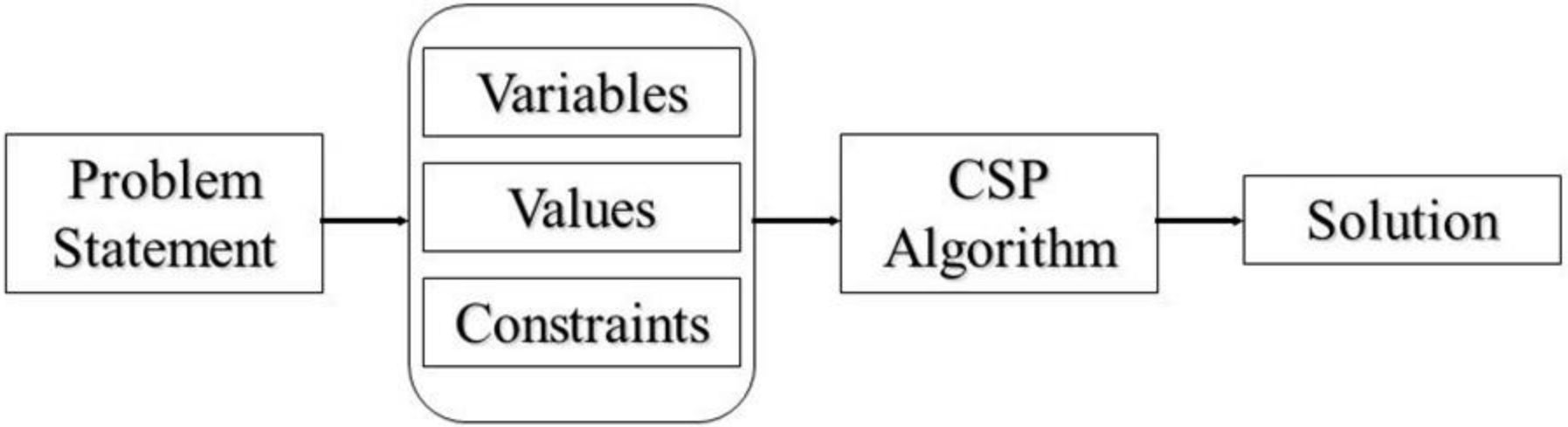
Constraint Graph

Search

X: red

Y: blue

Z: green



CSP as a standard search problem

- A CSP can easily expressed as a standard search problem.
- Incremental formulation
  - *Initial State*: the empty assignment {}.
  - *Successor function*: Assign value to an unassigned variable provided that there is no conflict.
  - *Goal test*: the current assignment is complete.
  - *Path cost*: as constant cost for every step.

CSP as a standard search problem

- This is the same for all CSP's !
- Solution is found at depth *n* (if there are *n* variables).
  - Hence depth first search can be used.
- Path is irrelevant.
- Branching factor *b* at the top level is *nd.*
- *b=(n-l)d* at depth *l*, hence $n!d^n$ leaves can be generated (but only max. d^n complete assignments, O(n^n), Stirling's approx., a very stupid way of searching, something better needed – heuristic functions?!).

Commutativity and finite depth

- CSPs are commutative.
  - The order of any given set of actions (value assignements) has no effect on the outcome.
  - Example: choose colors for Australian territories one at a time
    - [WA=red then NT=green] same as [NT=green then WA=red]
    - All practical CSP search algorithms consider **a single variable assignment at a time**, if $\#D = d$, $\#Var = n$ $\Rightarrow$ there are d^n leaves, at n depth.
- With a finite d number of variables the search tree is of finite depth.

Backtracking search

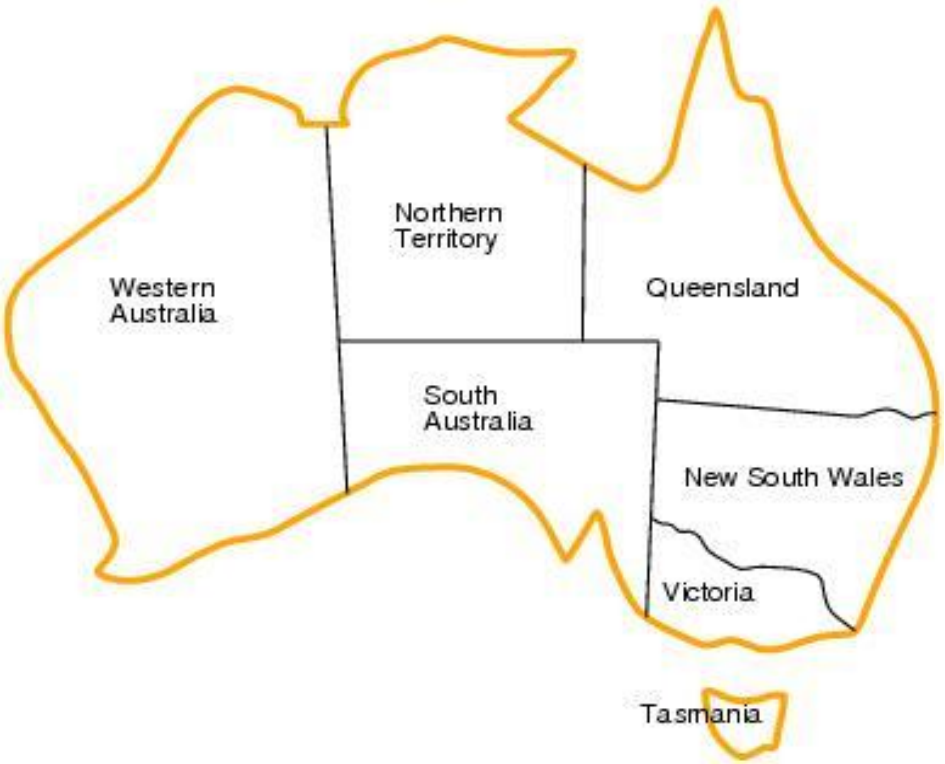
- Cfr. Depth-first search
- Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Uninformed algorithm
  - No good general performance (see table p. 143)

Backtracking search

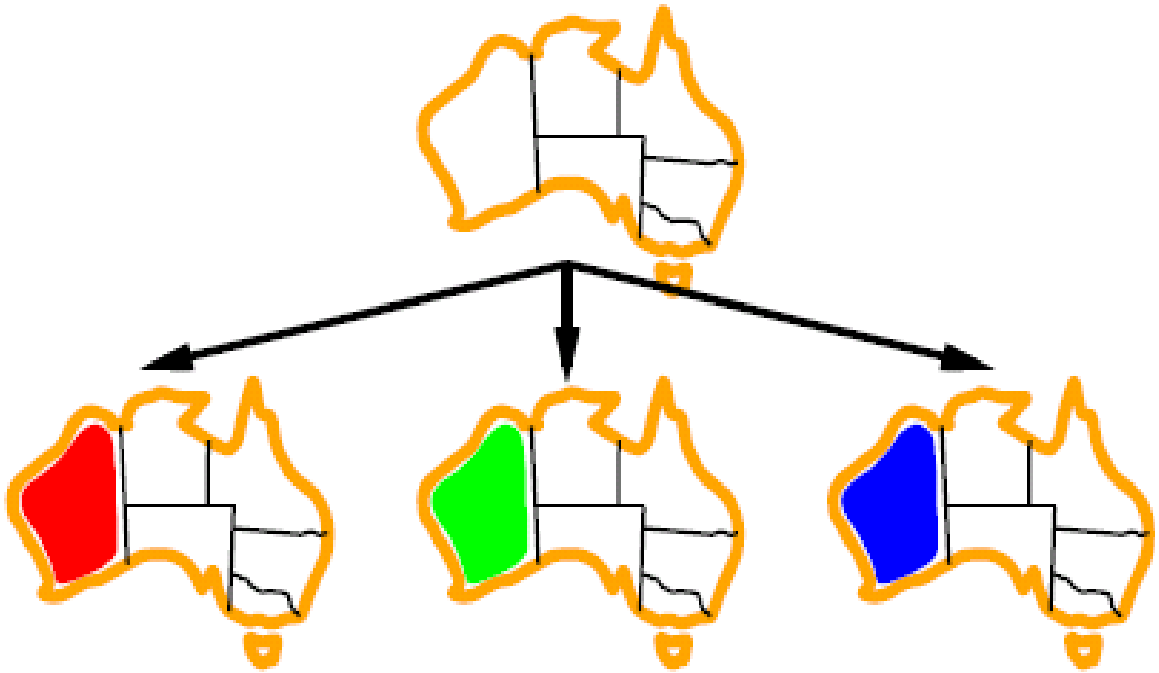
function BACKTRACKING-SEARCH(*csp*)
return a solution, or failure
  return RECURSIVE-BACKTRACKING(*{} , csp*)

function RECURSIVE-BACKTRACKING(*assignment, csp*)
return a solution, or failure
  if *assignment* is complete **then return** *assignment*
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
  for each *value* **in** ORDER-DOMAIN-VALUES(*var, assignment, csp*) **do**
    if *value* is consistent with *assignment* according to
    CONSTRAINTS[*csp*] **then**
      add *{var=value}* to assignment
      result ← RECURSIVE-BACTRACKING(*assignment, csp*)
      if *result* ≠ *failure* **then return** *result*
      remove *{var=value}* from *assignment*
  return *failure*

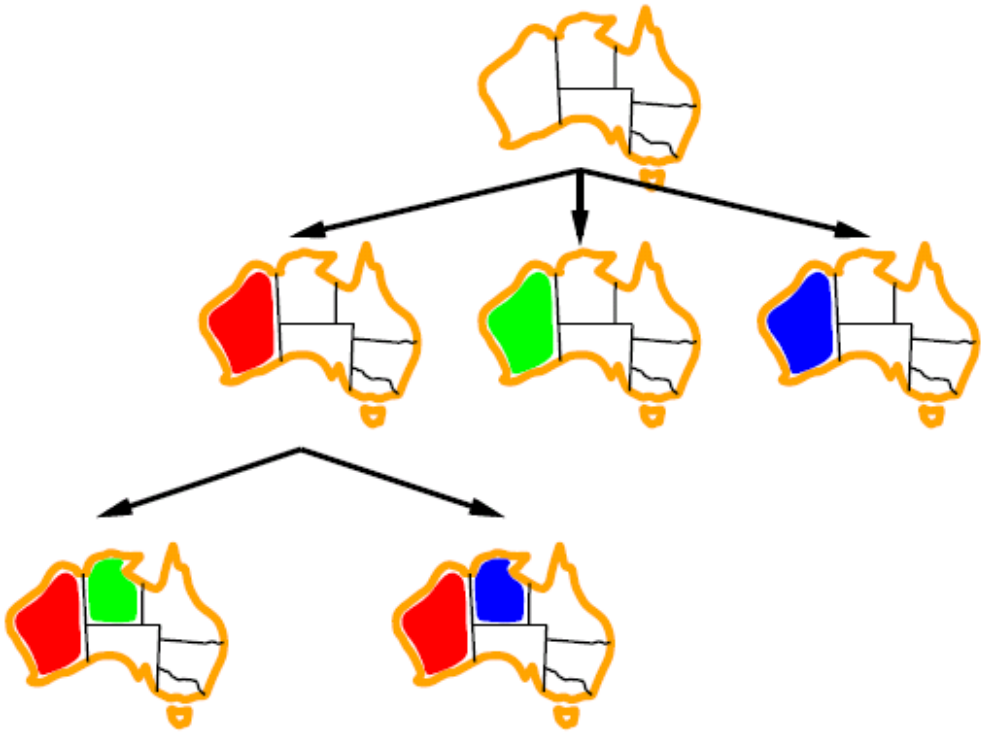
Backtracking search



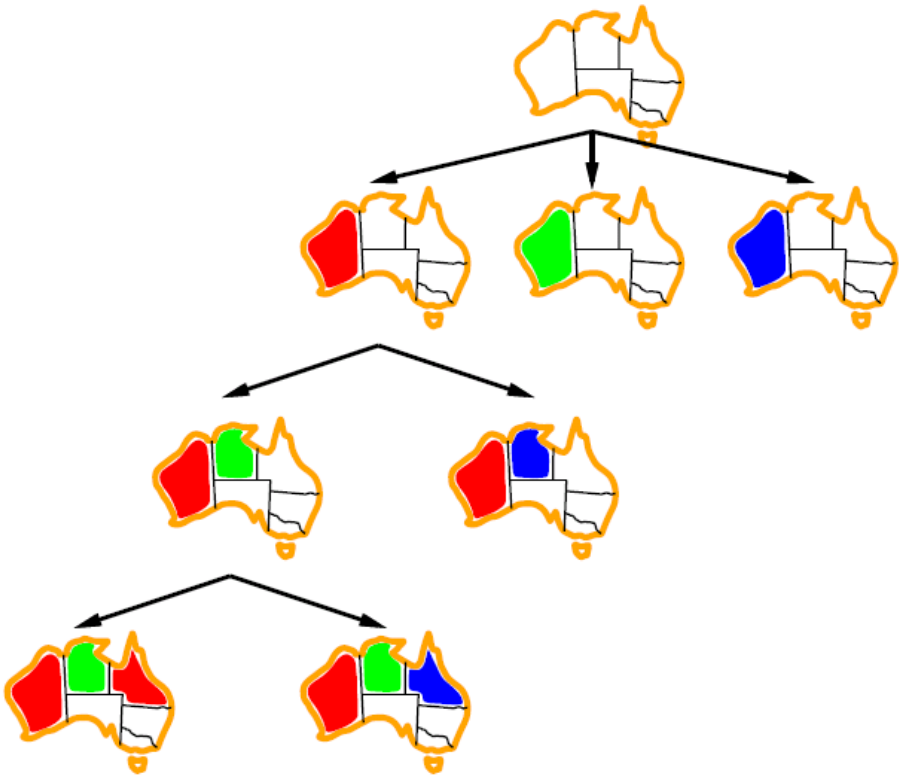
Backtracking search



Backtracking search



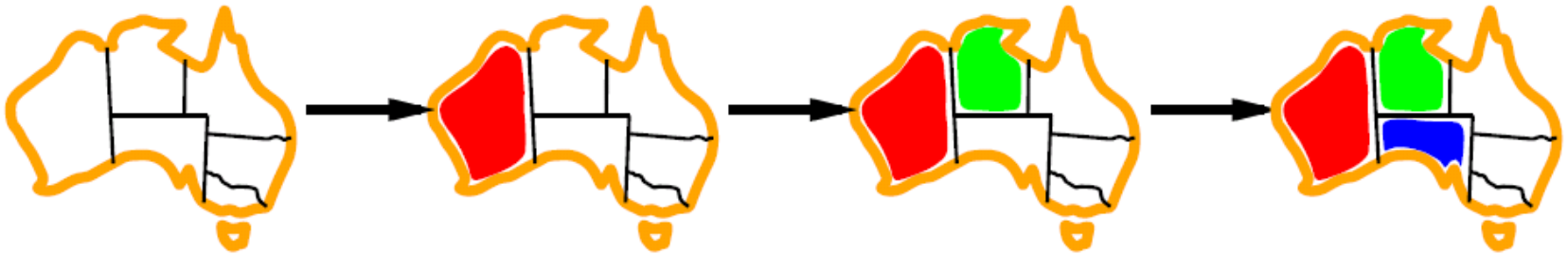
Backtracking search



Improving backtracking efficiency

- Want improvements? → introduce heuristics
  - Domain independent (general-purpose, ok!)
  - Domain dependent (how?)
- General-purpose methods can give huge gain in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?

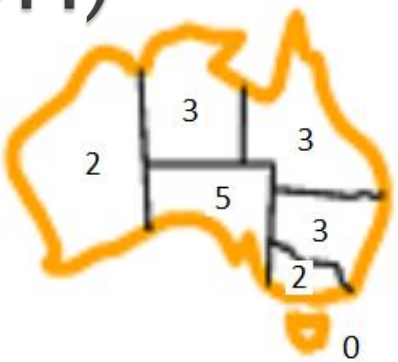
Most constraining variable (MCV) (Minimum remaining values)

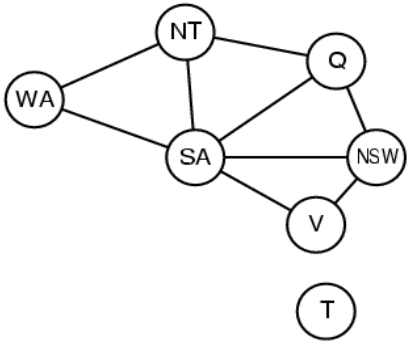


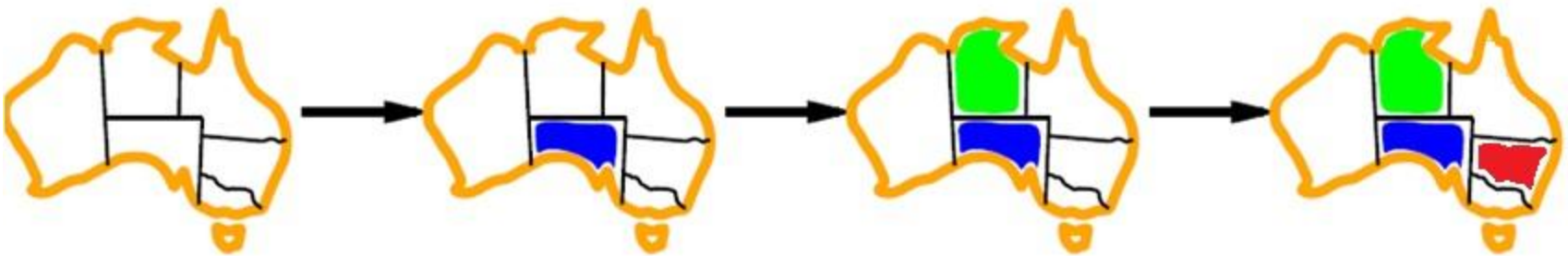
var ← SELECT-UNASSIGNEDVARIABLE(VARIABLES[*csp*],*assignment*,*csp*)

- A.k.a. most constrained variable heuristic („fail fast”)
- *Rule*: choose variable with the fewest legal values
- *Which variable shall we try first?*

Degree heuristic (DH)

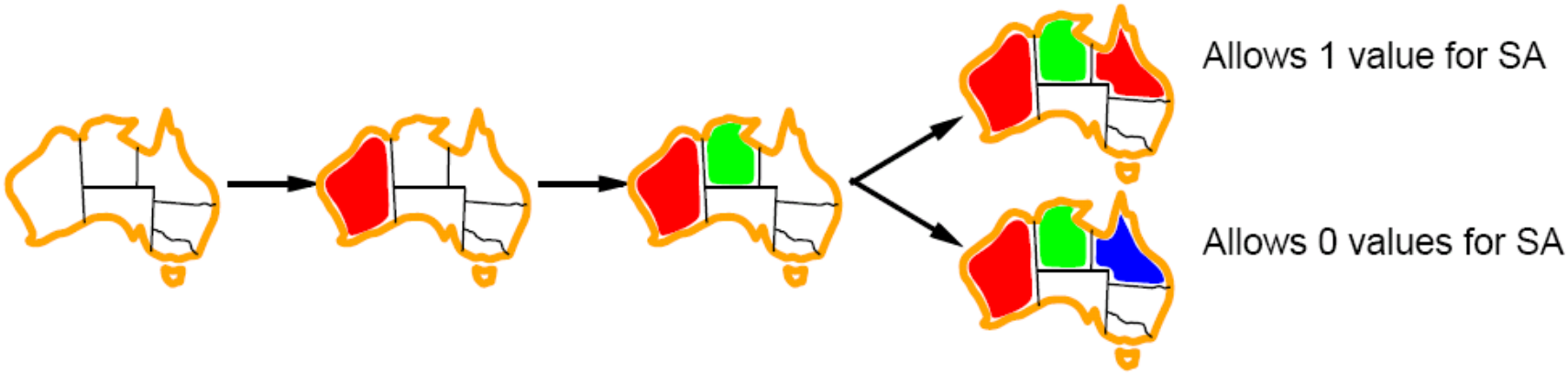






- Use degree heuristic
- *Rule*: select variable that is involved in the largest number of constraints on other unassigned variables.
- Degree heuristic is very useful as a tie breaker.
- *In what order should its values be tried?*

Least constraining value (LCV)



- Least constraining value heuristic
- *Rule*: given a variable, choose the least constraining value i.e. the one that leaves the maximum flexibility for subsequent variable assignments.

Forward checking

- Can we detect inevitable failure early?
  - *And avoid it later?*
- *Forward checking idea:* keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values.

Constraint propagation

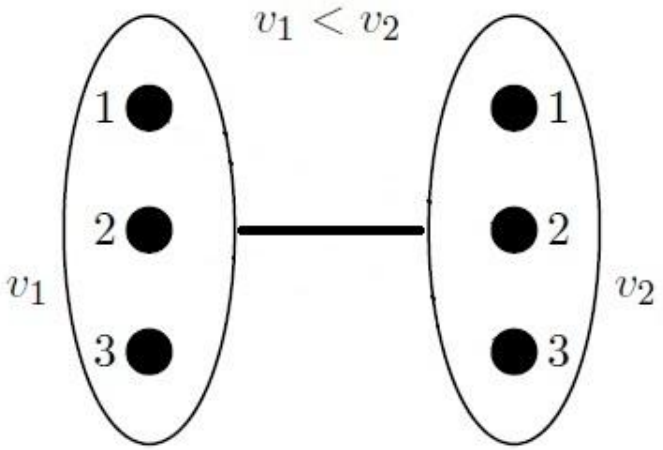
- Forward checking does not see failures in advance.
- NT and SA cannot be blue at the same time.

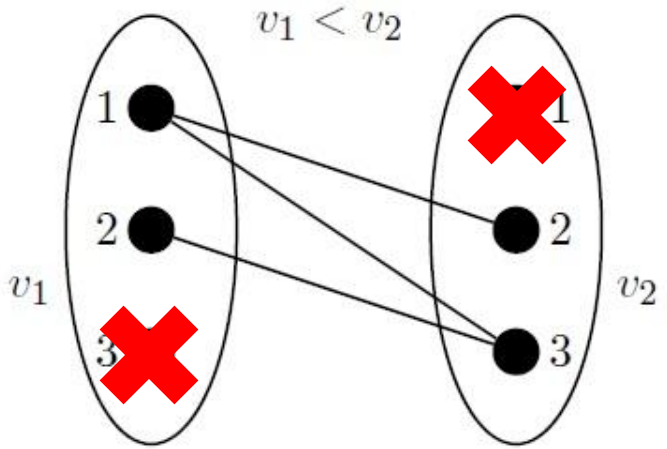
Arc-consistency

X → *Y* is **arc-consistent** if and only if for every value of *X* there exist some legal value of *Y*.

If X has lost a legal value, its neighbours must be check anew.

Arc-consistency checking can be used as a pre-processing step, before starting the search, or during the search, at every moment when a new value is assigned to a variable.

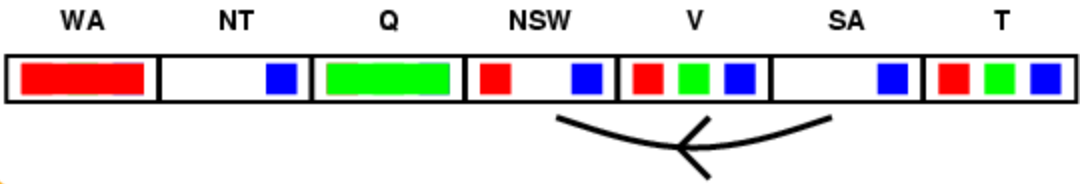


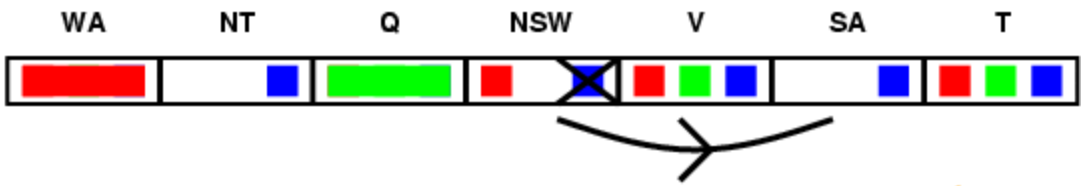


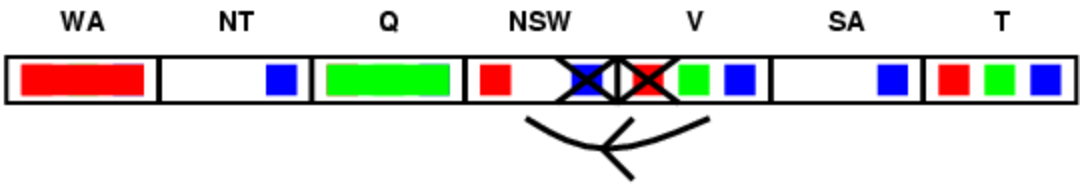
K-consistency

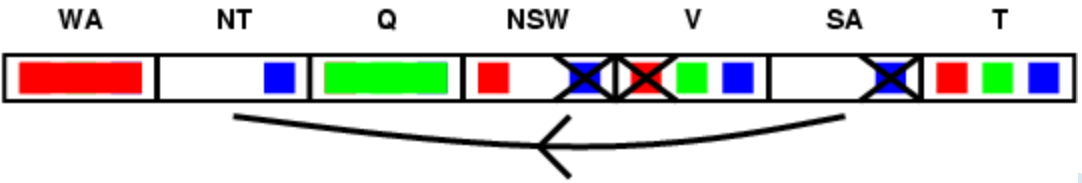
- A CSP is *k*-consistent if for any set of *k*-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any *k*th variable.
- A graph is strongly *k*-consistent if
  - It is *k*-consistent and
  - Is also (*k*-1) consistent, (*k*-2) consistent, … all the way down to 1-consistent.
- YET *no free lunch*: any algorithm for establishing *n*-consistency must take time exponential in *n*, in the worst case.
- (1-consistency = node-consistency (unary constraints)
- 2-consistency = arc-consistency (binary constraints)
- 3-consistency = path-consistency) (…)

Arc-consistency









Local search (optimization) for CSP

- Use complete-state representation
- For CSPs
  - allow states with unsatisfied constraints
  - operators **reassign** variable values
- Variable selection: randomly select any conflicted variable
- Value selection: *min-conflicts heuristic*
  - Select new value that results in a minimum number of conflicts with the other variables

Local search for CSP

function MIN-CONFLICTS(*csp, max_steps*) **return** solution or failure

inputs: *csp*, a constraint satisfaction problem

max_steps, the number of steps allowed before giving up

current ← an initial complete assignment for *csp*

for *i* = 1 to *max_steps* **do**

if *current* is a solution for *csp* then return *current*

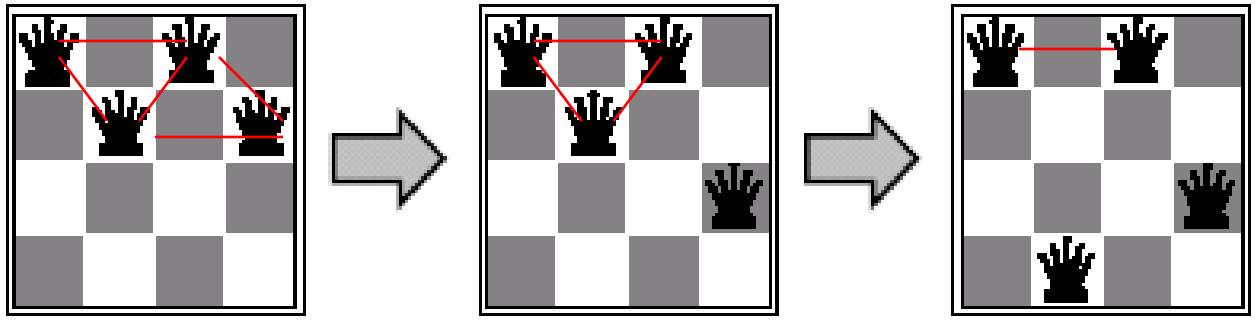
var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]

value ← the value *v* for *var* that minimizes CONFLICTS(*var,v,current,csp*)

set *var* = *value* in *current*

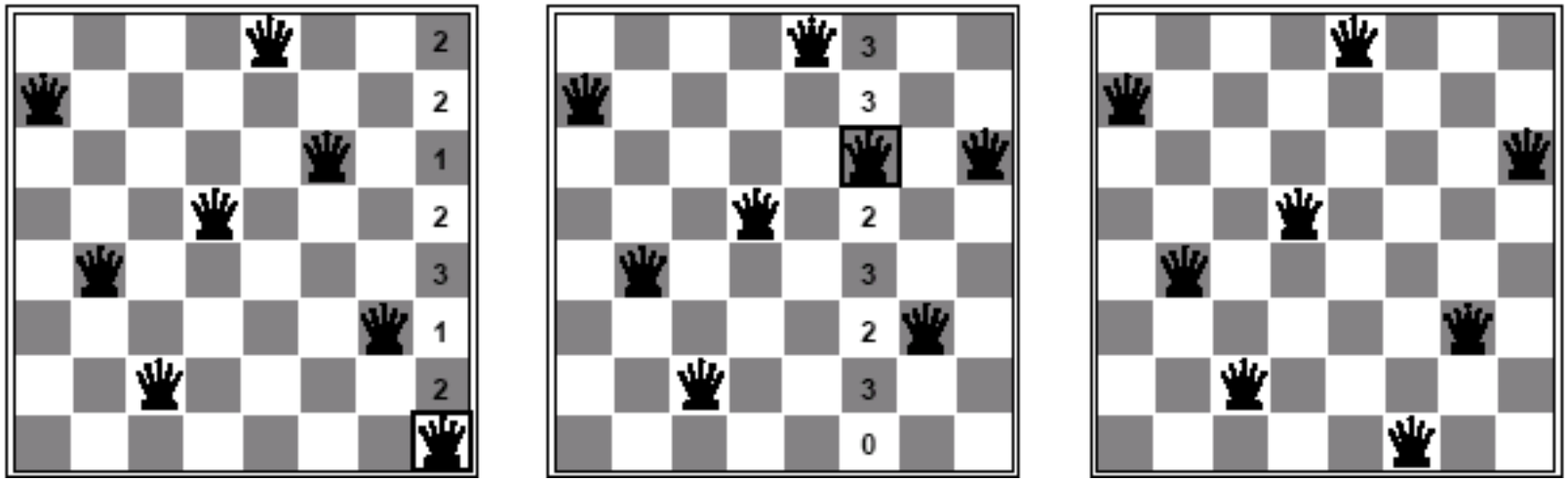
return *faiilure*

Min-conflicts example 1



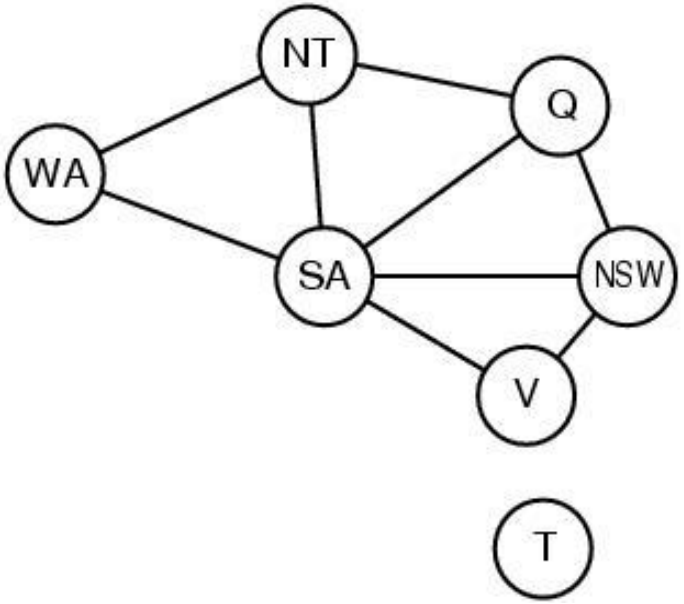
- Use of min-conflicts heuristic in hill-climbing.

Min-conflicts example 2



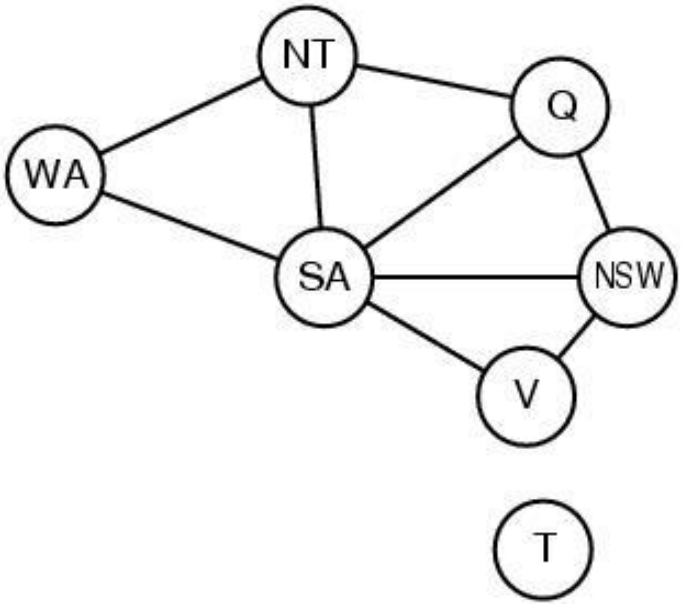
- A two-step solution for a given 8-queens problem using min-conflicts heuristic.
- At each stage a queen is chosen for reassignment in its column.
- The algorithm moves the queen to the min-conflict square breaking ties randomly.

Problem structure



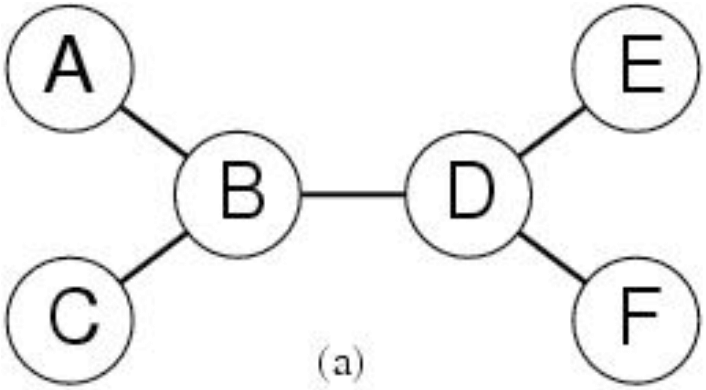
- *How can the problem structure help to find a solution quickly?*
- Subproblem identification is important:
  - E.g. coloring Tasmania and mainland are independent subproblems
  - Identifiable as connected components of constrained graph.
- Improves performance

Problem structure



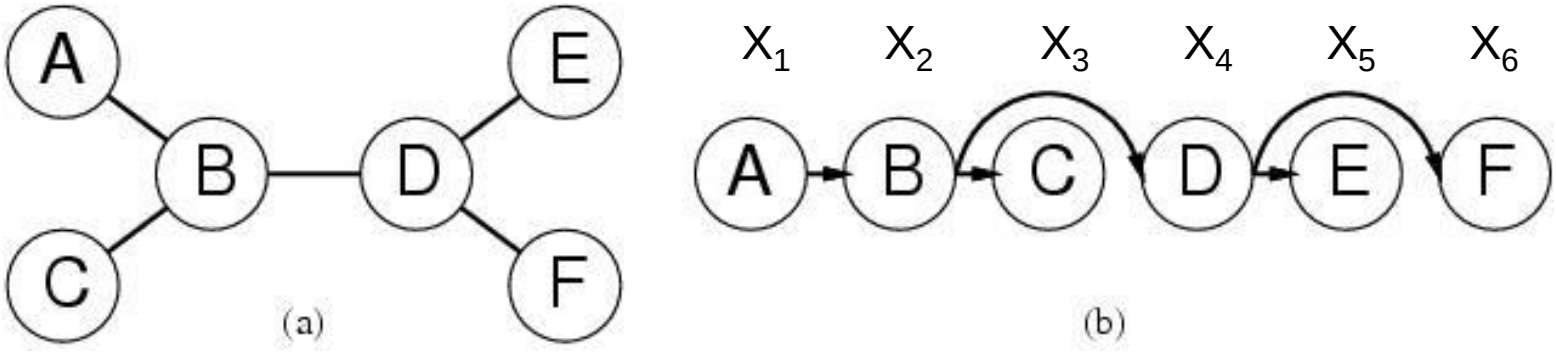
- Suppose each problem has
  c variables out of a total of *n*.
- Worst case solution cost is $O((n/c) \times d^c)$,
  i.e. linear in *n*
  - Instead of $O(d^n)$, exponential in *n!*
- E.g. *n= 80, c= 20, d=2*
  - 2^{80} = 4 billion years at 1 million nodes/sec.
  - $4 * 2^{20}$= 0.4 second at 1 million nodes/sec

Tree-structured CSP



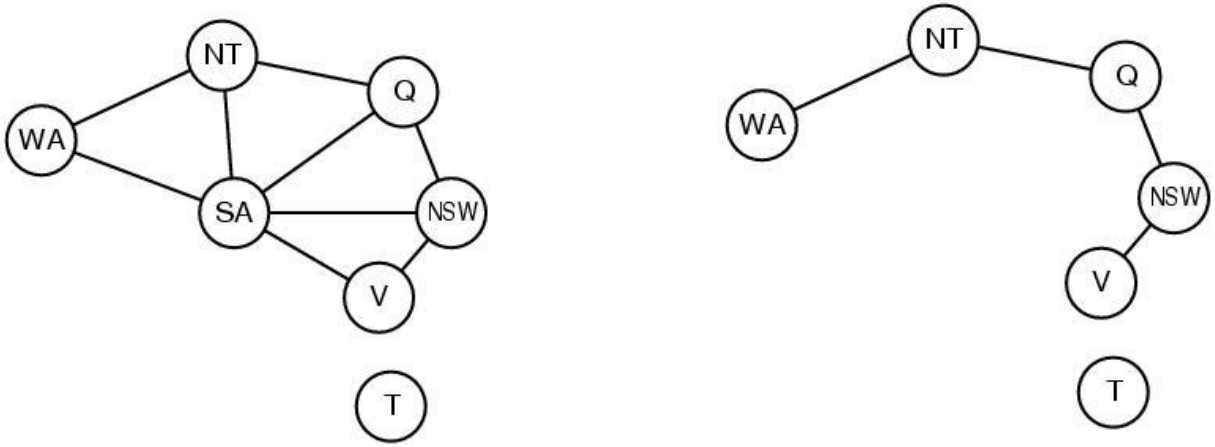
- Theorem: if the constraint graph has no loops then CSP can be solved in $O(nd^2)$ time
  (linear in the number of variables!)
- Compare difference with general CSP, where worst case is $O(d^n)$

Tree-structured CSPs



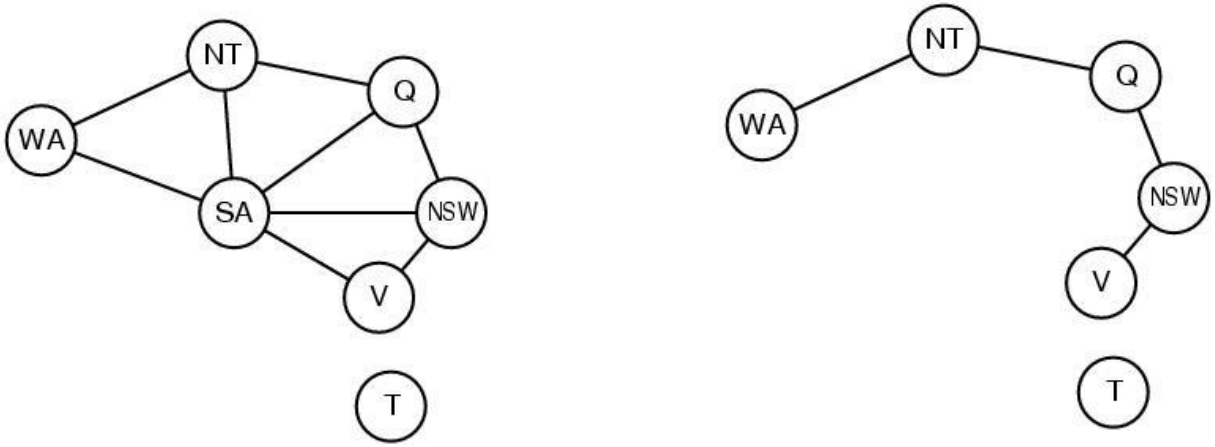
- In most cases subproblems of a CSP are connected as a tree
- Any tree-structured CSP can be solved in time linear in the number of variables.
  - Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering.
  - For *j* from *n* down to 2, apply REMOVE-INCONSISTENT-VALUES(Parent(X_j),X_j)
  - For *j* from 1 to *n* assign X_j consistently with Parent(X_j)

Nearly tree-structured CSP



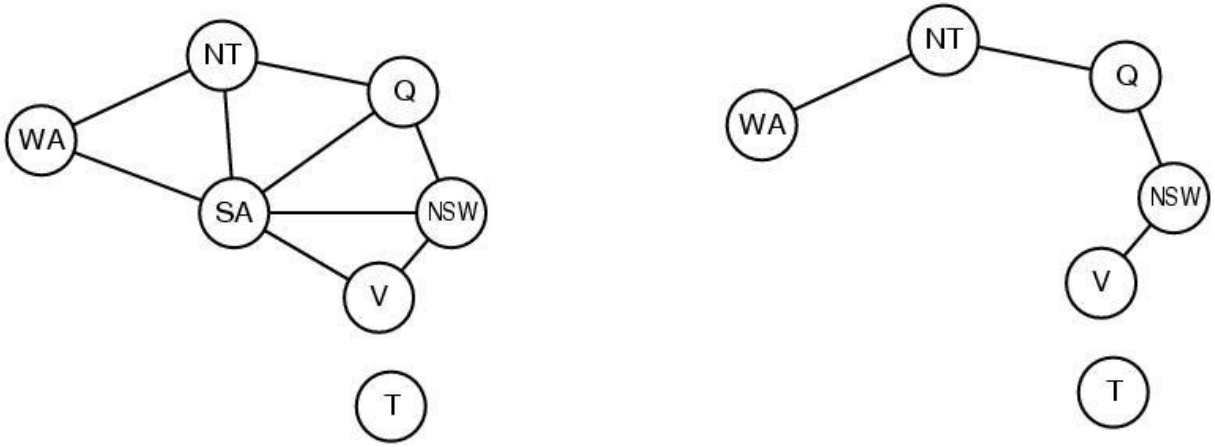
- *Can a more general constraint graph be reduced to tree?*
- Two approaches:
  - Remove certain nodes
  - Collapse certain nodes

Nearly tree-structured CSP



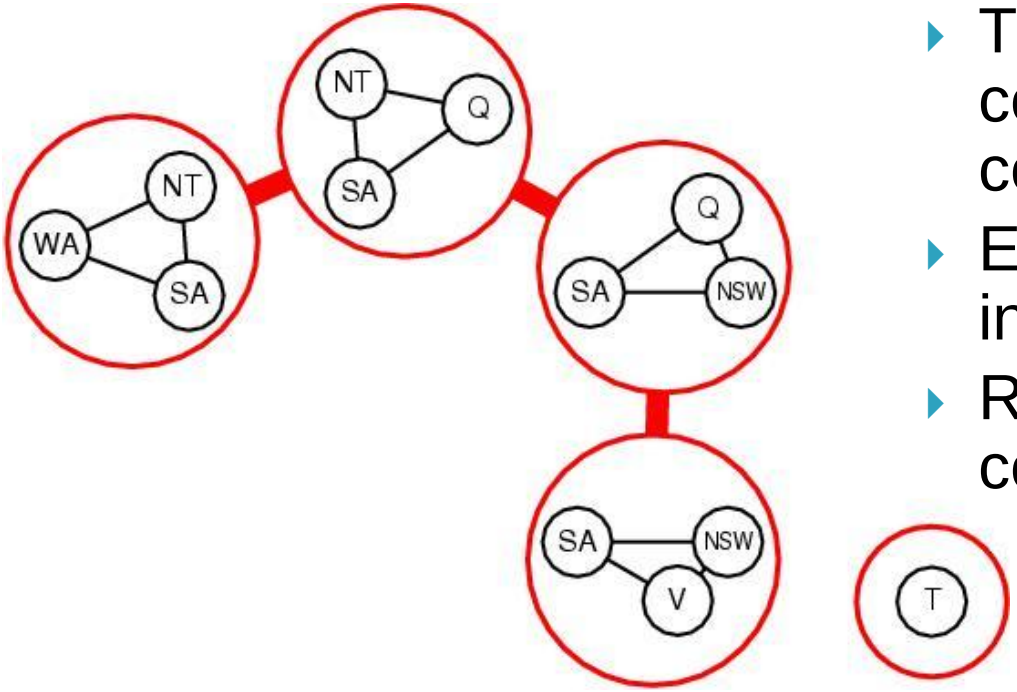
- Idea: assign values to some variables so that the remaining variables form a tree.
- Assume that we assign *{SA=x}* ← *cycle cutset*
  - And remove any values from the other variables that are inconsistent.
  - The selected value for SA could be wrong so we have to try all of them.

Nearly tree-structured CSP



- This approach is worthwhile if cycle cutset is small.
- Finding the smallest cycle cutset is NP-hard
  - Approximation algorithms exist
- This approach is called *cutset conditioning*.

Nearly tree-structured CSP



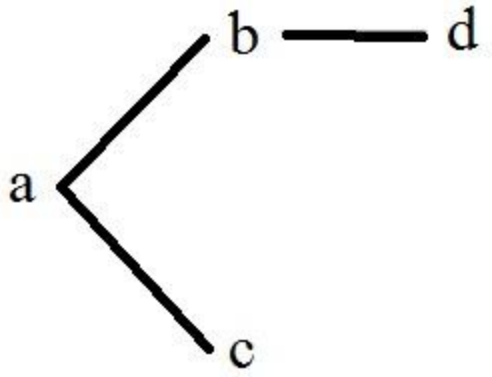
- Tree decomposition of the constraint graph into a set of connected subproblems.
- Each subproblem is solved independently
- Resulting solutions are combined.

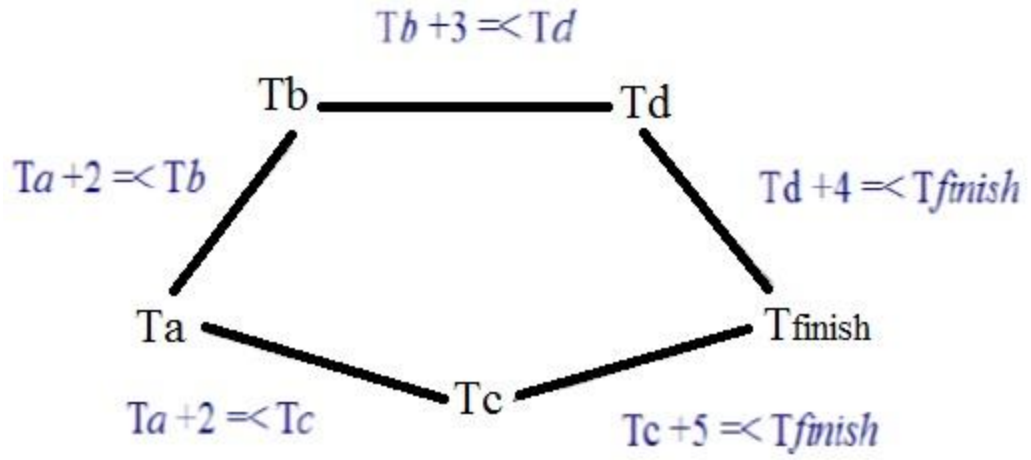
- Necessary requirements:
  - Every variable appears in at least one of the subproblems.
  - If two variables are connected in the original problem, they must appear together in at least one subproblem.
  - If a variable appears in two subproblems, it must appear in each node on the path.

Summary

- CSPs are a special kind of problem: states defined by values of a fixed set of variables, goal test defined by constraints on variable values
- Backtracking=depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that lead to failure.
- Constraint propagation does additional work to constrain values and detect inconsistencies.
- The CSP representation allows analysis of problem structure.
- Tree structured CSPs can be solved in linear time.
- Iterative min-conflicts is usually effective in practice.

Examples





1. $Ta + 2 \leq Tb$
2. $Ta + 2 \leq Tc$
3. $Tb + 3 \leq Td$
4. $Tc + 5 \leq Tfinish$
5. $Td + 4 \leq Tfinish$

LTa = 2h, LTb = 3h, LTc = 5h, LTd = 4h
Ta, Tb, Tc, Td, Tf = (0,1,2,3,4,5,6,7,8,9,10)

The earliest finishing time (start of T_{finish})?
(Solution: Ta = 0, Tb = 2, Tc = (2,4), Td = 5, T_{finish} = 9)

Examples

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

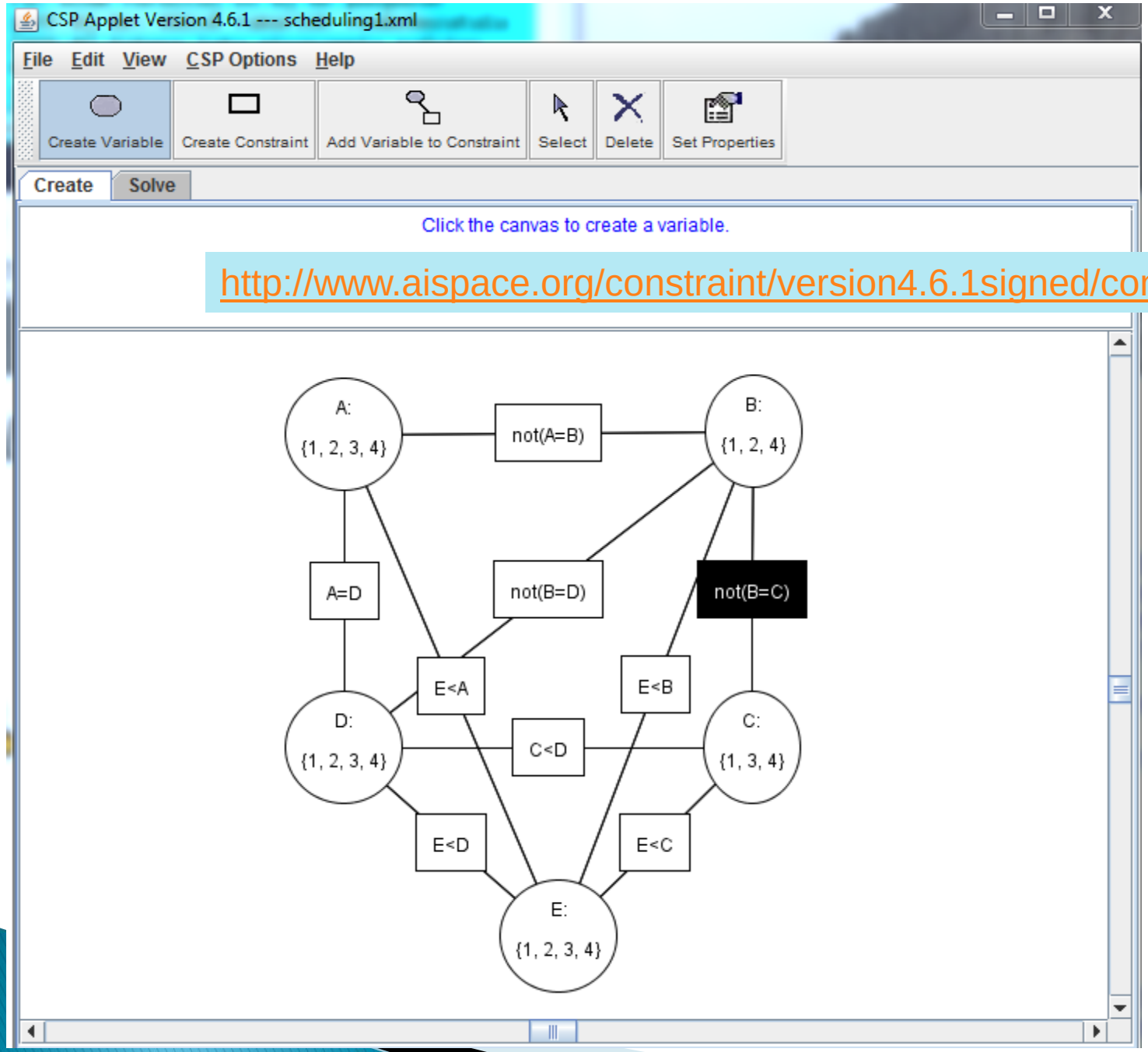
$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

$C_1 : a \vee b$
$C_2 : c \vee d$
$C_3 : a \vee e \vee f$
$C_4 : \neg b \vee \neg f \vee g$
$C_5 : \neg f \vee h$
$C_6 : \neg b \vee \neg h \vee i$
$C_7 : \neg g \vee \neg i$

a = 0
b = 1
c = 0
d = 1
e = 0
f = 1, g = 1, i = 0, h = 1
X
e = 1



http://www.aispace.org/constraint/version4.6.1signed/constraint.jnlp