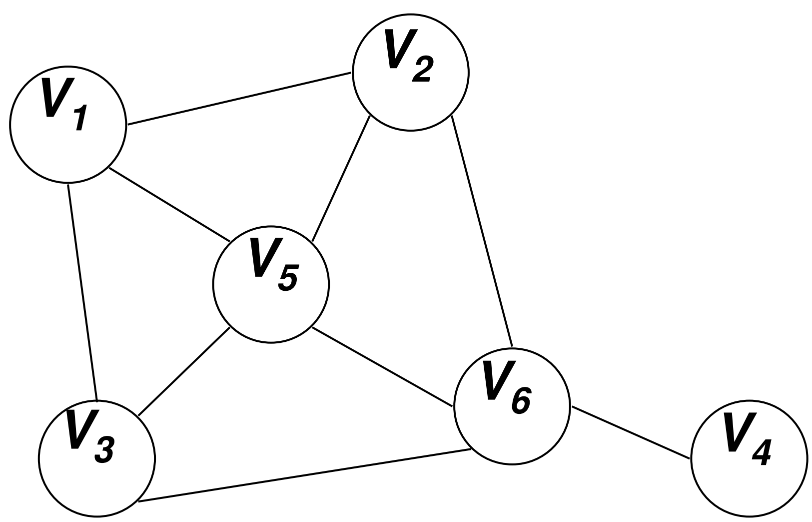
Constraint Satisfaction Problems

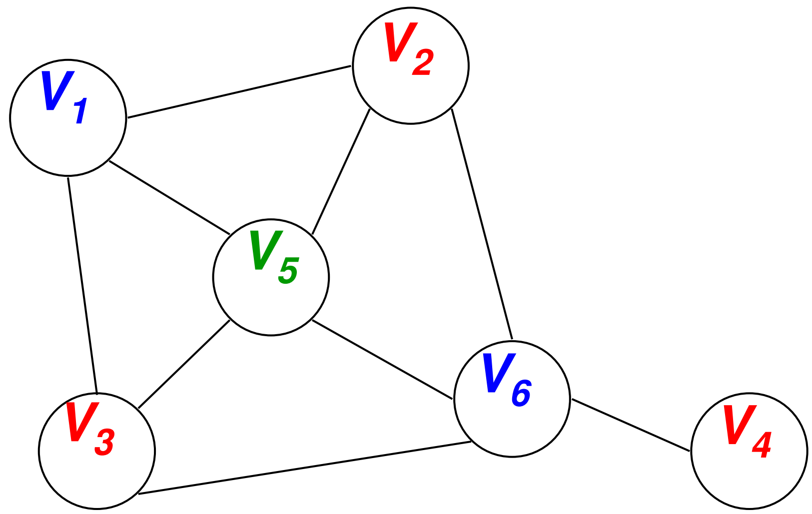
R&N Chapter 5

Animations from http://www.cs.cmu.edu/~awm/animations/constraint

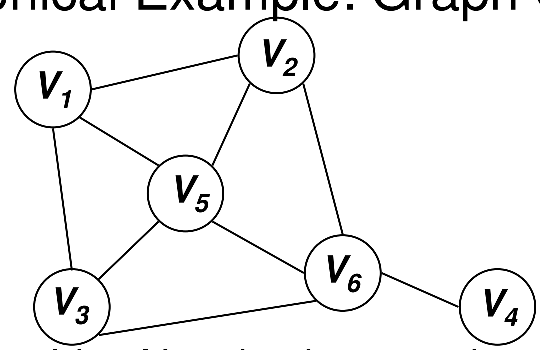
Outline

- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems



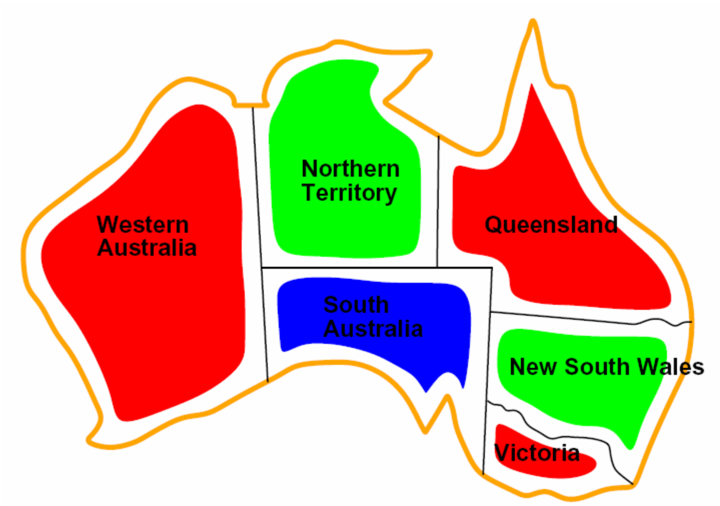


Canonical Example: Graph Coloring



- Consider *N* nodes in a graph
- Assign values $V_1,..,V_N$ to each of the *N* nodes
- The values are taken in {*R*,*G*,*B*}
- Constraints: If there is an edge between *i* and *j*, then V_i must be different of V_j

Canonical Example: Graph Coloring



CSP Definition

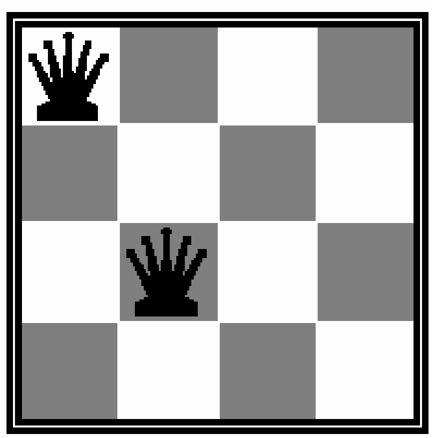
- CSP = {V, D, C}
- *Variables*: $V = \{V_1,..,V_N\}$
  - Example: The values of the nodes in the graph
- *Domain*: The set of d values that each variable can take
  - Example: $D = \{R, G, B\}$
- *Constraints*: $C = \{C_1,..,C_K\}$
- Each constraint consists of a tuple of variables and a list of values that the tuple is allowed to take for this problem
  - Example: $[(V_2,V_3),\{(R,B),(R,G),(B,R),(B,G),(G,R),(G,B)\}]$
- Constraints are usually defined implicitly → A function is defined to test if a tuple of variables satisfies the constraint
  - Example: $V_i \neq V_j$ for every edge (i,j)

Binary CSP

- Variable ***V*** and ***V'*** are connected if they appear in a constraint
- Neighbors of ***V*** = variables that are connected to ***V***
- The domain of ***V***, *D*(***V***), is the set of candidate values for variable ***V***
- $D_i = D(V_i)$

- Constraint graph for binary CSP problem:
  - Nodes are variables
  - Links represent the constraints
  - Same as our canonical graph-coloring problem

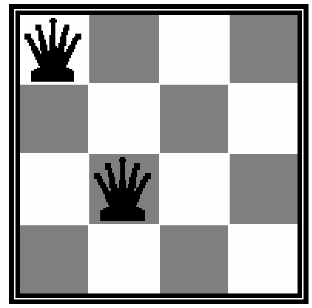
N-Queens



$$Q_1 = 1 \quad Q_2 = 3$$

Example: N-Queens

- Variables: Q_i
- Domains: $D_i = \{1, 2, 3, 4\}$
- Constraints
  - $Q_i \neq Q_j$ (cannot be in same row)
  - $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)



$Q_1 = 1 \quad Q_2 = 3$

- Valid values for (Q_1, Q_2) are (1,3) (1,4) (2,4) (3,1) (4,1) (4,2)

Cryptarithmetic

$$
\begin{array}{r}
S\,E\,N\,D \\
+\,M\,O\,R\,E \\
\hline
M\,O\,N\,E\,Y
\end{array}
$$

Example: Cryptarithmetic

- Variables
  D, E, M, N, O, R, S, Y
- Domains
  {0, 1, 2, 3 ,4, 5, 6, 7, 8, 9 }
- Constraints
  $M \neq 0$, $S \neq 0$ (unary constraints)
  $Y = D + E$ OR $Y = D + E - 10$.
  $D \neq E$, $D \neq M$, $D \neq N$, etc.

```
  S E N D
+ M O R E
---------
M O N E Y
```

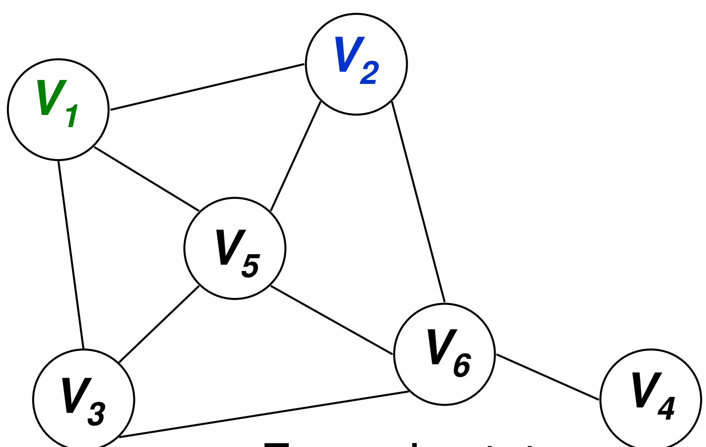
More Useful Examples

- Scheduling
- Product design
- Asset allocation
- Circuit design
- Constrained robot planning
- ....

Outline

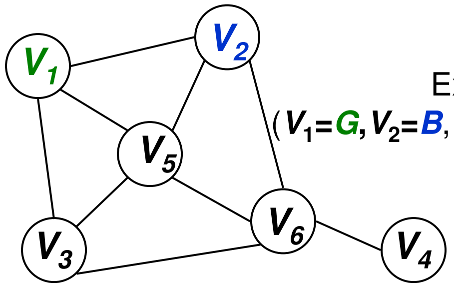
- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems

Search Space



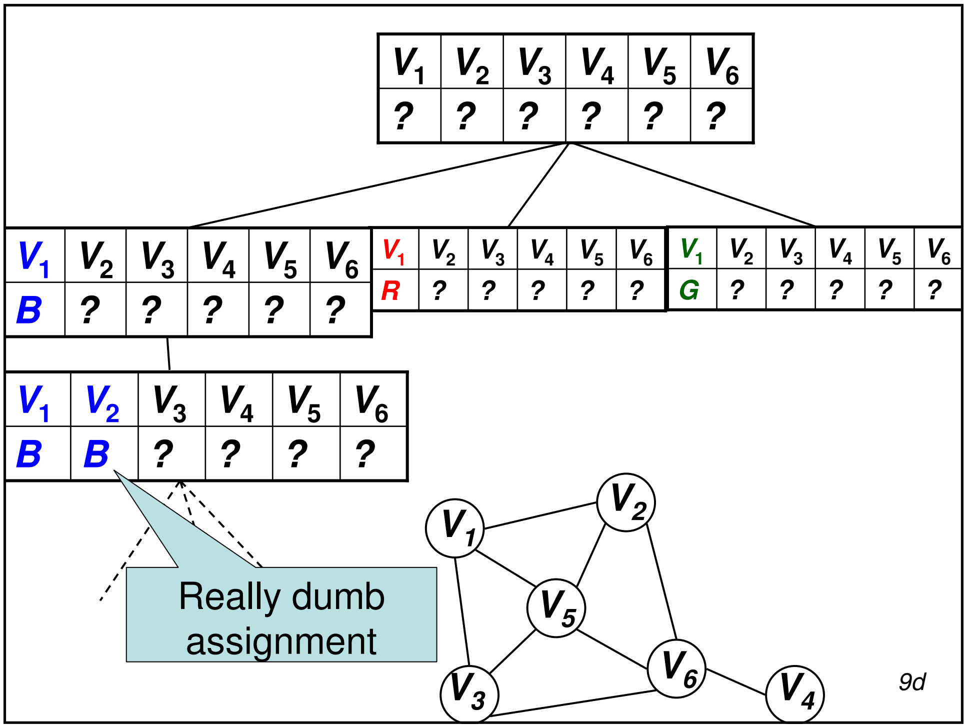
Example state:
(V_1=G, V_2=B, V_3=?, V_4=?, V_5=?, V_6=?)

Search Space



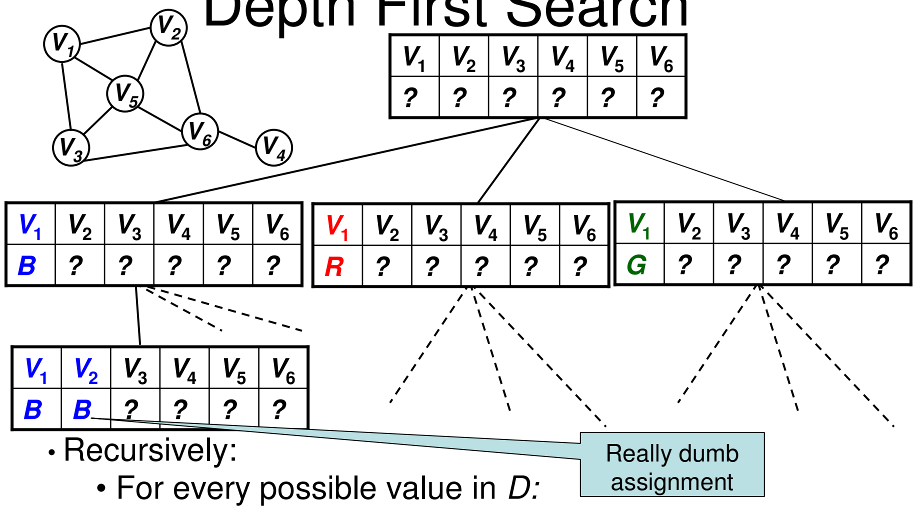
Example state:
(V_1=*G*, V_2=*B*, V_3=?, V_4=?, V_5=?, V_6=?)

- *State*: assignment to *k* variables with *k*+1,..,*N* unassigned
- *Successor*: The successor of a state is obtained by assigning a value to variable *k*+1, keeping the others unchanged
- *Start state*: (V_1=?, V_2=?, V_3=?, V_4=?, V_5=?, V_6=?)
- *Goal state*: All variables assigned with constraints satisfied
- No concept of cost on transition → We just want to find a solution, we don't worry how we get there



9d

Depth First Search



- Recursively:
  - For every possible value in *D:*
    - Set the next unassigned variable in the successor to that value
    - Evaluate the successor of the current state with this variable assignment
    - Stop as soon as a solution is found

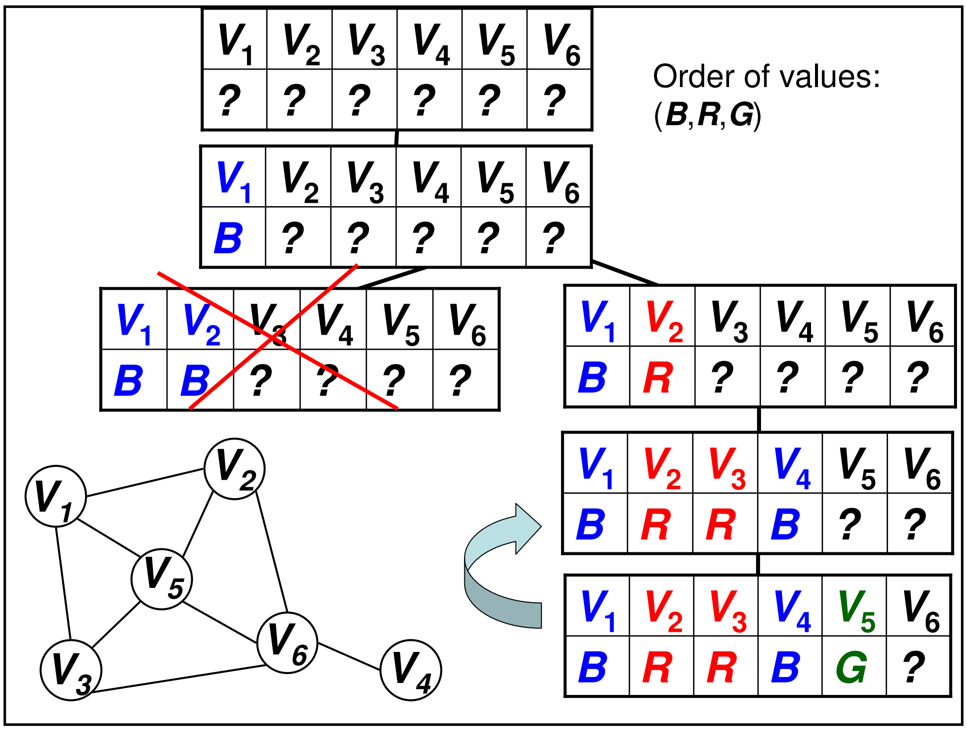
9d

DFS

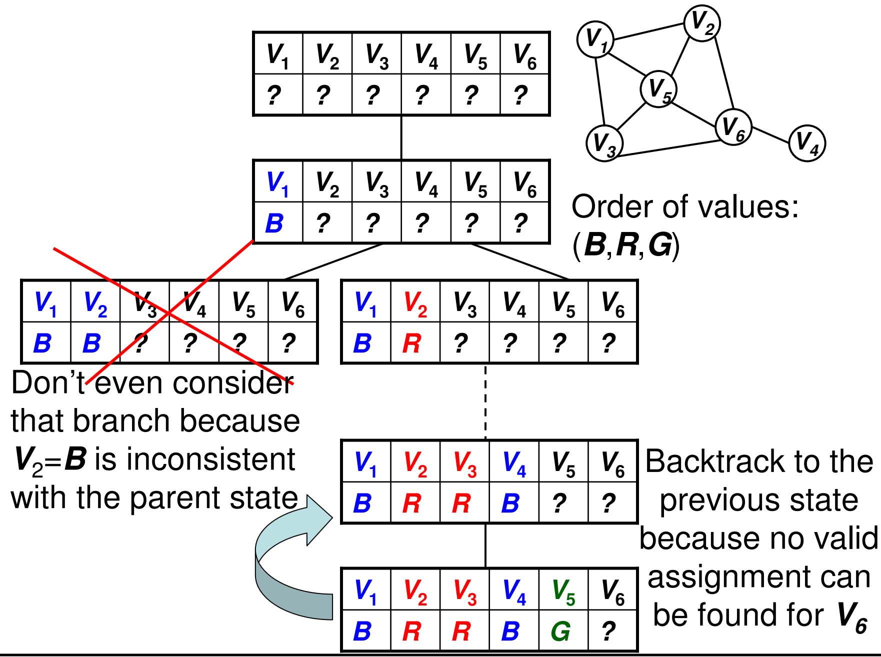
- Improvements:
  - Evaluate only value assignments that do not violate any constraints with the current assignments
  - Don't search branches that obviously cannot lead to a solution
  - Predict valid assignments ahead
  - Control order of variables and values

Outline

- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems



Backtracking DFS



Backtracking DFS

- For every possible value *x* in *D:*
  - If assigning *x* to the next unassigned variable V_{k+1} does not violate any constraint with the *k* already assigned variables:
    - Set the variable V_{k+1} to *x*
    - Evaluate the successors of the current state with this variable assignment
- If no valid assignment is found: Backtrack to previous state
- Stop as soon as a solution is found

9b, 27b

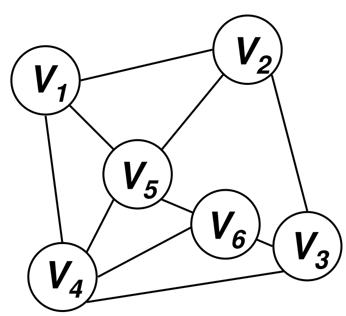
Backtracking DFS Comments

- Additional computation: At each step, we need to evaluate the constraints associated with the current candidate assignment (variable, value).
- Uninformed search, we can improve by predicting:
  – What is the effect of assigning a variable on all of the other variables?
  – Which variable should be assigned next and in which order should the values be evaluated?
  – When a branch fails, how can we avoid repeating the same mistake?

Forward Checking

- Keep track of remaining legal values for unassigned variables
- Backtrack when any variable has no legal values

	V_1	V_2	V_3	V_4	V_5	V_6
R	?	?	?	?	?	?
B	?	?	?	?	?	?
G	?	?	?	?	?	?

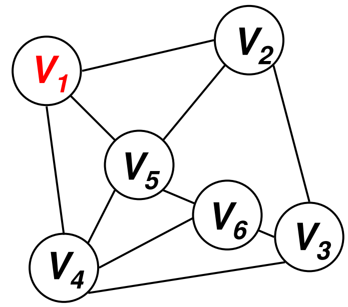


Warning: Different example with order (R,B,G)

Forward Checking

- Keep track of remaining legal values for unassigned variables
- Backtrack when any variable has no legal values

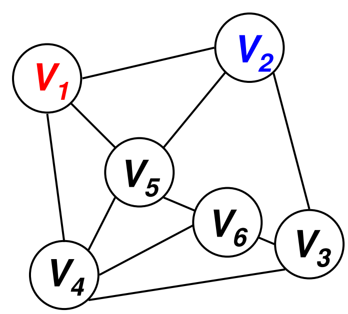
	V_1	V_2	V_3	V_4	V_5	V_6
R	O	X	?	X	X	?
B		?	?	?	?	?
G		?	?	?	?	?



Forward Checking

- Keep track of remaining legal values for unassigned variables
- Backtrack when any variable has no legal values

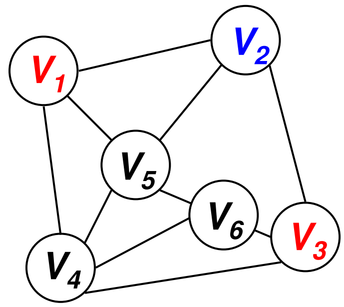
	V_1	V_2	V_3	V_4	V_5	V_6
R	O		?	X	X	?
B		O	X	?	X	?
G			?	?	?	?



Forward Checking

- Keep track of remaining legal values for unassigned variables
- Backtrack when no variable has a legal value

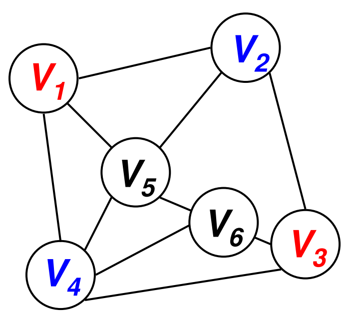
	V_1	V_2	V_3	V_4	V_5	V_6
R	O		O	X	X	X
B		O		?	X	?
G				?	?	?



Forward Checking

- Keep track of remaining legal values for unassigned variables
- Backtrack when any variable has no legal values

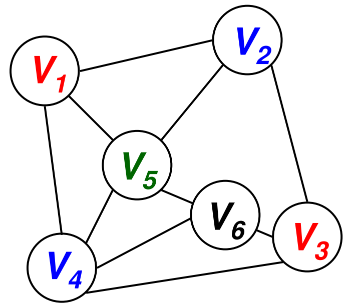
	V_1	V_2	V_3	V_4	V_5	V_6
R	O		O		X	X
B		O		O	X	X
G					?	?



Forward Checking

- Keep track of remaining legal values for unassigned variables
- Backtrack when any variable has no legal values

	V_1	V_2	V_3	V_4	V_5	V_6
R	O		O			X
B		O		O		X
G					O	X



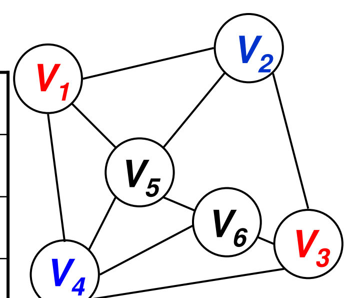
There are no valid assignments left for V_6 we need to backtrack

27f

Constraint Propagation

- Forward checking does not detect all the inconsistencies, only those that can be detected by looking at the constraints which contain the current variable.
- Can we look ahead further?

	V_1	V_2	V_3	V_4	V_5	V_6
R	O		O		X	X
B		O		O	X	X
G					?	?



At this point, it is already obvious that this branch will not lead to a solution because there are no consistent values in the remaining domain for V_5 and V_6.

Constraint Propagation

- ***V*** = variable being assigned at the current level of the search
- Set variable ***V*** to a value in *D*(***V***)
- For every variable ***V'*** connected to ***V***:
  - Remove the values in *D*(***V'***) that are inconsistent with the assigned variables
  - For every variable ***V''*** connected to ***V'***:
    - Remove the values in *D*(***V''***) that are no longer possible candidates
    - And do this again with the variables connected to ***V''***
      - ……..until no more values can be discarded

Constraint Propagation

New: Constraint Propagation

Forward Checking as before

- ***V*** = variable being assigned at the current level of the search
- Set variable ***V*** to a value in *D*(***V***)
- For every variable ***V'*** connected to ***V***:
  - Remove the values in *D*(***V'***) that are inconsistent with the assigned variables
  - For every variable ***V''*** connected to ***V'***:
    - Remove the values in *D*(***V''***) that are no longer possible candidates
    - And do this again with the variables connected to ***V''***
      - ……..until no more values can be discarded

CP for the graph coloring problem

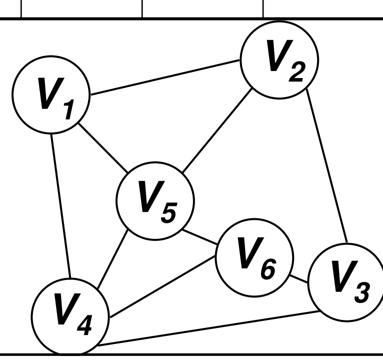
Propagate (*node*, *color*)

1. Remove color from the domain of all of the neighbors
2. For every neighbor *N*:

   If *D*(*N*) was reduced to only one color after step 1 (*D*(*N*) = {*c*}):

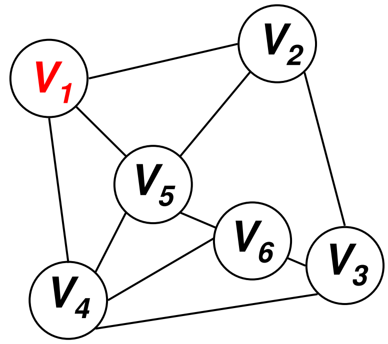
   Propagate (*N*,*c*)

	V_1	V_2	V_3	V_4	V_5	V_6
R	O	X	X	X	X	?
B		O	X	?	X	X
G		?	?	X	?	X



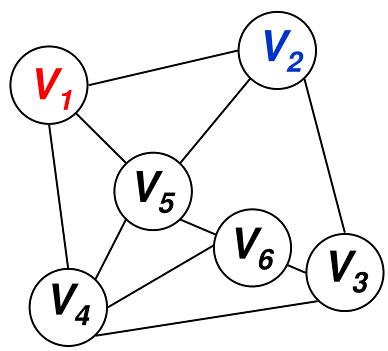
After Propagate *(V_1, R)*:

	V_1	V_2	V_3	V_4	V_5	V_6
R	O	X	?	X	X	?
B		?	?	?	?	?
G		?	?	?	?	?

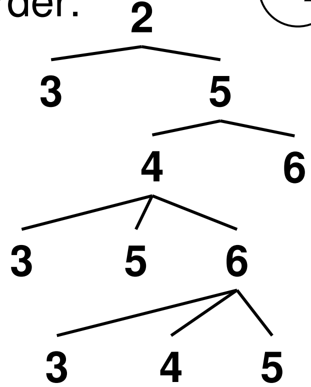


After Propagate *(V_2, B)*:

	V_1	V_2	V_3	V_4	V_5	V_6
R	O		X	X	X	?
B		O	X	?	X	X
G			?	X	?	X

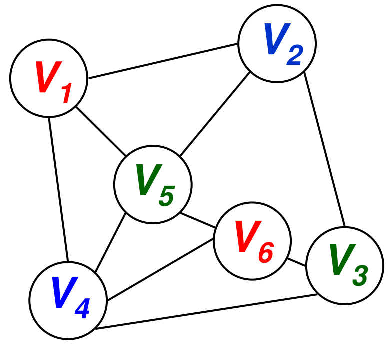


Propagation order:



After Propagate *(V_2, **B**)*:

	V_1	V_2	V_3	V_4	V_5	V_6
R	***O***		***X***	***X***	***X***	***?***
B		***O***	***X***	***?***	***X***	***X***
G			***?***	***X***	***?***	***X***



Note: We get directly to a solution in *one step of CP* after setting V_2 *without any additional search*

Some problems can even be solved by applying CP directly without search (if we're lucky)

More General CP: Arc Consistency

- *A* = queue of active arcs (V_i,V_j)
- Repeat while *A* not empty:
  - (V_i,V_j) ← next element of *A*
  - For each *x* in *D*(V_i):
    - Remove *x* from *D*(V_i) if there is no *y* in *D*(V_j) for which (*x*,*y*) satisfies the constraint between V_i and V_j.
  - If *D*(V_i) has changed:
    - Add all the pairs (V_k,V_i), where V_k is a neighbor of V_i (*k* not equal to *j*) to *A*

More General: *k*-Consistency

- Check consistency of sets of *k* variables instead of pairs of variables (arc consistency)
- Trade-off:
  - CP time increases rapidly with *k*
  - Search time may decrease with *k* (but maybe not as fast)
- Complete constraint propagation exponential in size of the problem

Outline

- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems

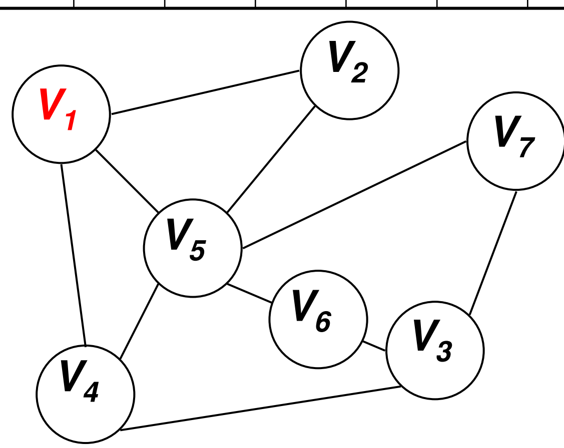
Variable and Value Heuristics

- So far we have selected the next variable and the next value by using a fixed order

1. Is there a better way to pick the next variable?
2. Is there a better way to select the next value to assign to the current variable?

CSP Heuristics: Variable Ordering I

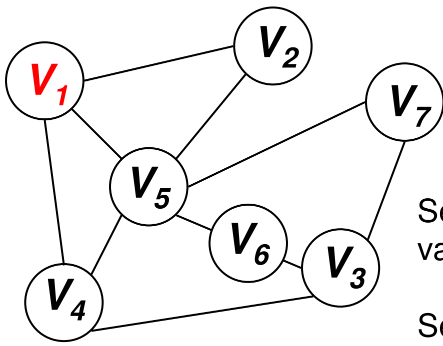
V_1	V_2	V_3	V_4	V_5	V_6	V_7
R	?	?	?	?	?	?



196v

CSP Heuristics: Variable Ordering I

- *Most Constraining Variable*
- Selecting a variable which contributes to the *largest* number of constraints will have the largest effect on the other variables → Hopefully will prune a larger part of the search
- This amounts to finding the variable that is connected to the largest number of variables in the constraint graph.



V_1	V_2	V_3	V_4	V_5	V_6	V_7
R	?	?	?	?	?	?

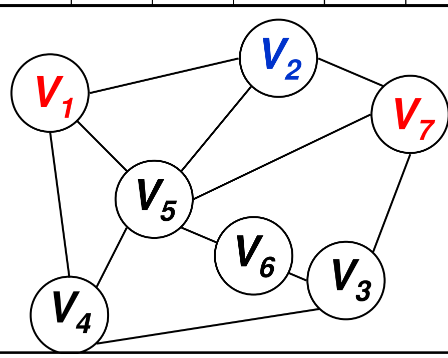
Setting variable V_5 affects *4* variables

Setting variable V_2 (or V_3, V_4) affects fewer variables

196v

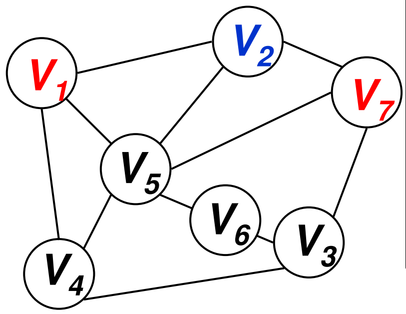
CSP Heuristics: Variable Ordering II

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
R	O		X	X	X	?	O
B		O	?	?	X	?	
G			?	?	?	?	



CSP Heuristics: Variable Ordering II

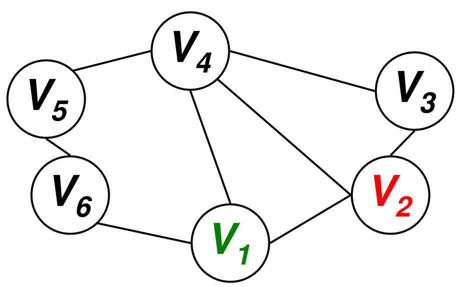
- *Minimum Remaining Values (MRV)*
- Selecting the variable that has the least number of candidate values is most likely to cause a failure early ("fail-first" heuristic)



	V_1	V_2	V_3	V_4	V_5	V_6	V_7
R	O		X	X	X	?	O
B		O	?	?	X	?	
G			?	?	?	?	

V_5 is the most constrained variable and is the most likely to prune the search tree

CSP Heuristics: Value Ordering



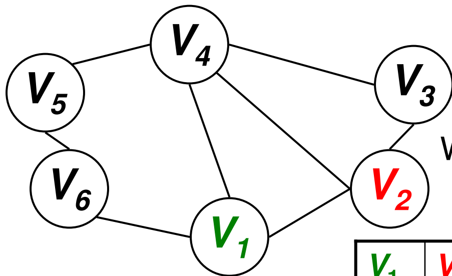
Four colors:
$D = \{\mathbf{R}, \mathbf{G}, \mathbf{B}, \mathbf{Y}\}$

V_1	V_2	V_3	V_4	V_5	V_6	V_7
G	R	?	?	?	?	?

Warning: Different example!!!

CSP Heuristics: Value Ordering

- *Least Constraining Value*
- Choose the value which causes the smallest reduction in the number of available values for the neighboring variables



Four colors: $D = \{\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}, \boldsymbol{Y}\}$

Which value to try next for V_3?

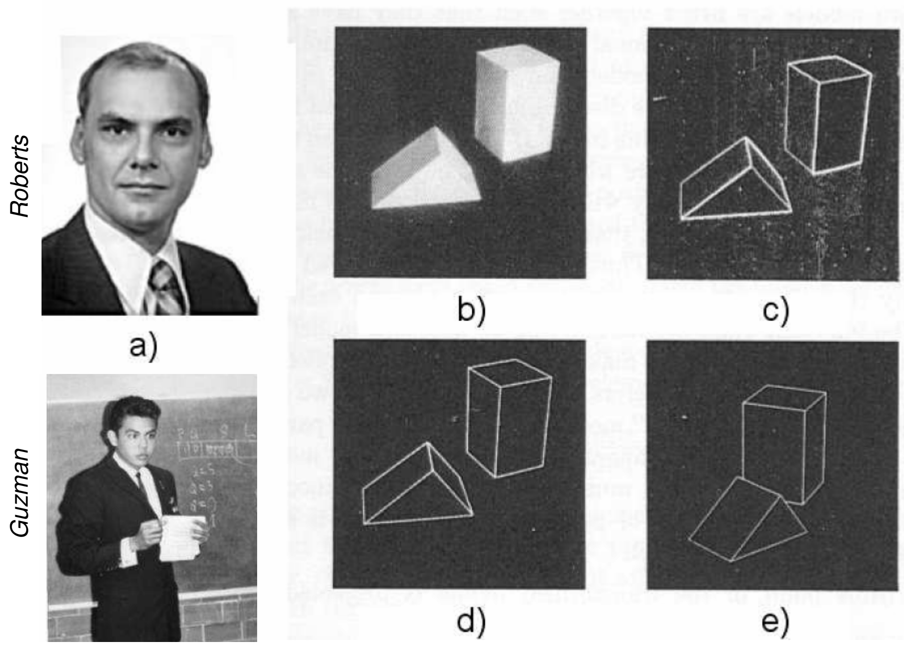
V_1	V_2	V_3	V_4	V_5	V_6	V_7
G	***R***	***?***	***?***	***?***	***?***	***?***

Warning: Different example!!!

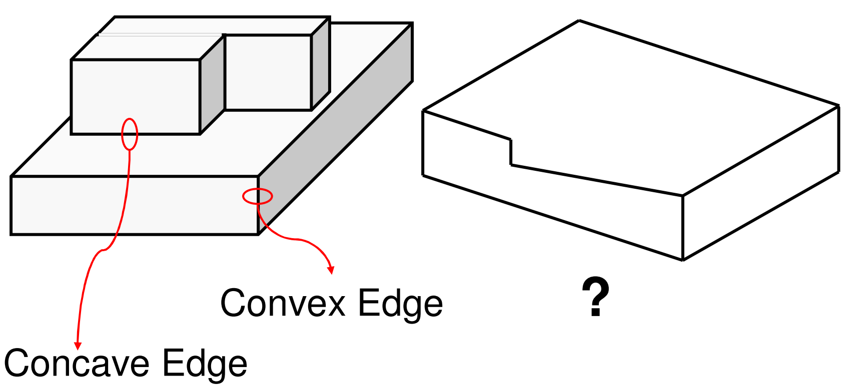
Outline

- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems

1964-70

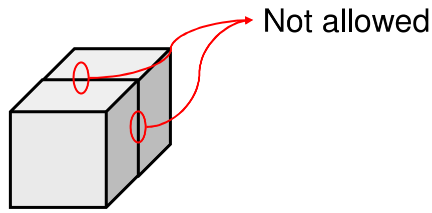


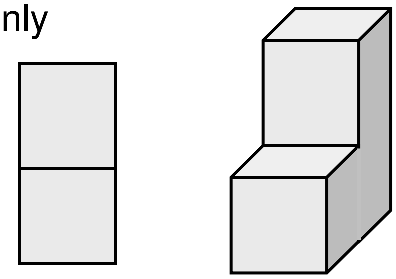
CP Example:Line Drawing Interpretation



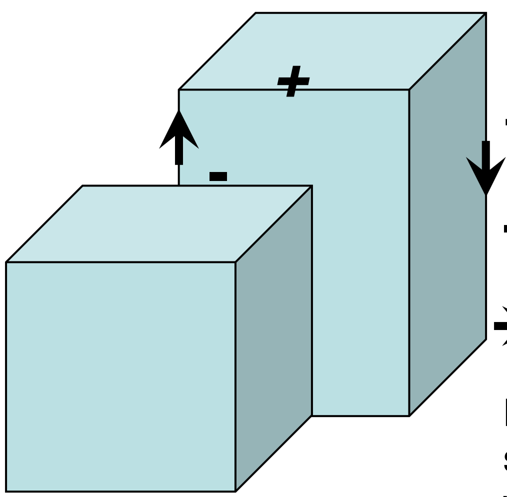
Assumptions

- No shadows
- No edge between common planes
- General viewpoint
- Trihedral corners only





3 Possible Edge Labels



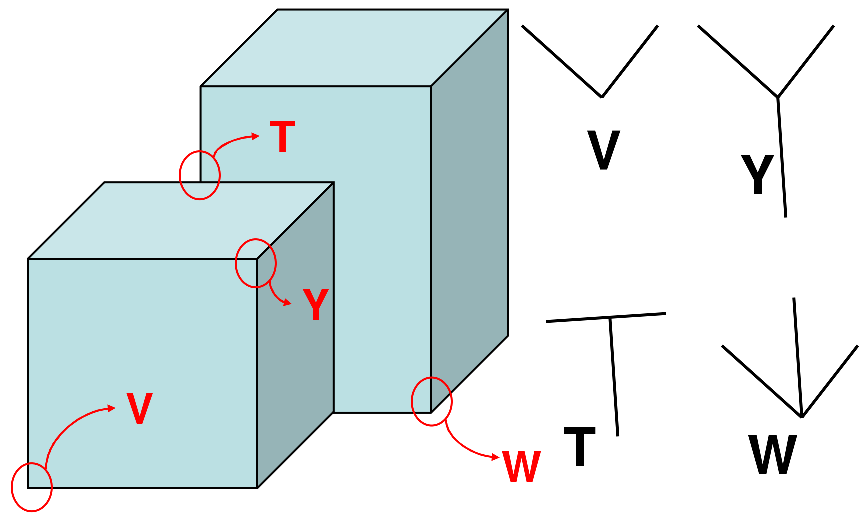
+ : Convex edge

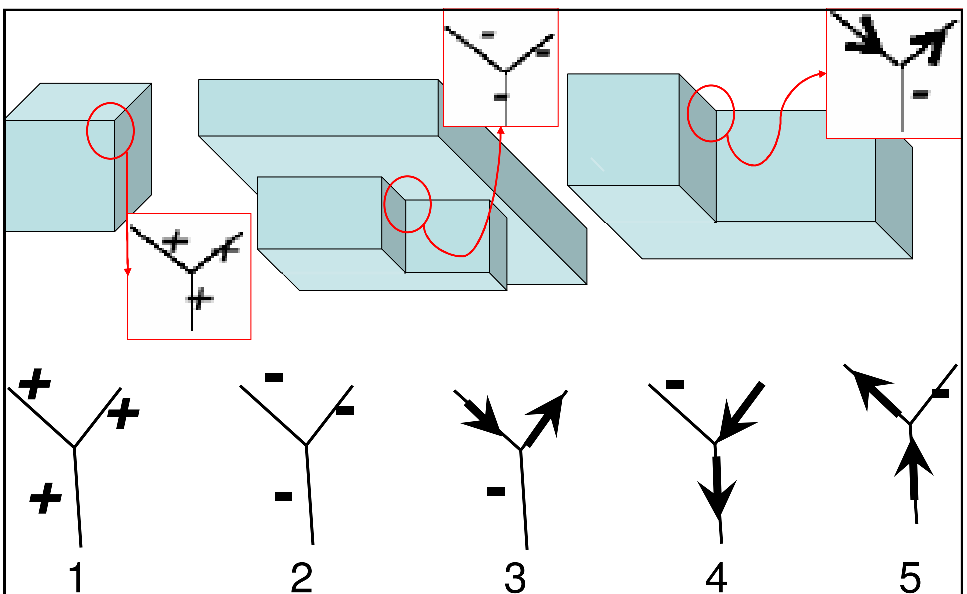
- : Concave edge

➔ : Boundary edge

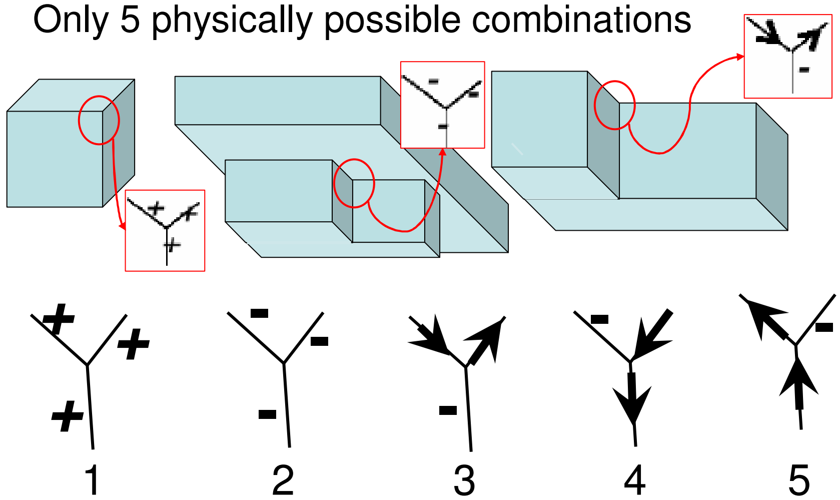
By convention: the surface is to the right when looking in the direction of the arrow

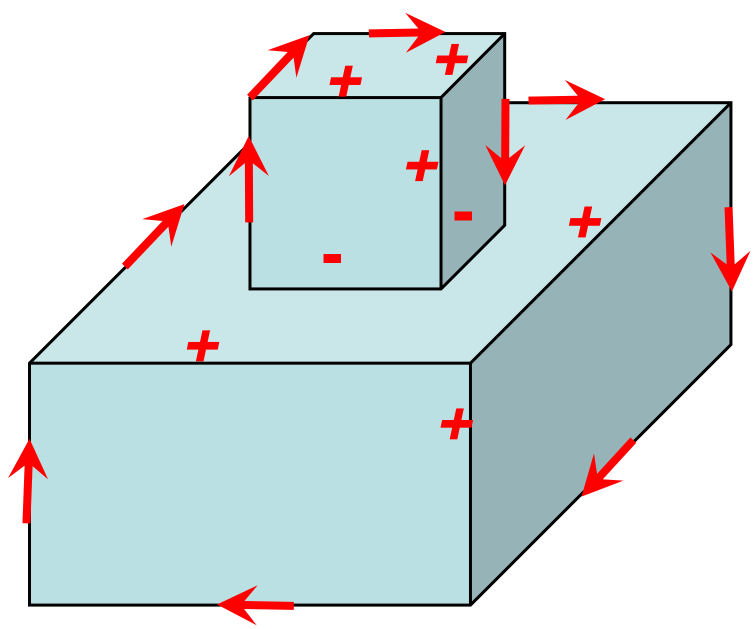
4 Possible Types of Junctions





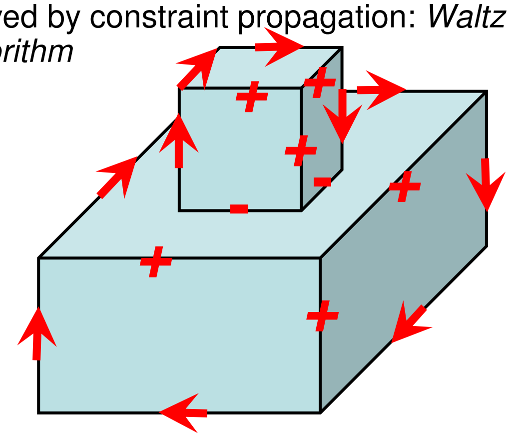
There are 3 x $4^3 + 4^2 = 208$ possible combination of edge labels and junctions types
For example, 4^3 possible combinations of labels at a ***Y*** junction, but…
Only 5 physically possible combinations





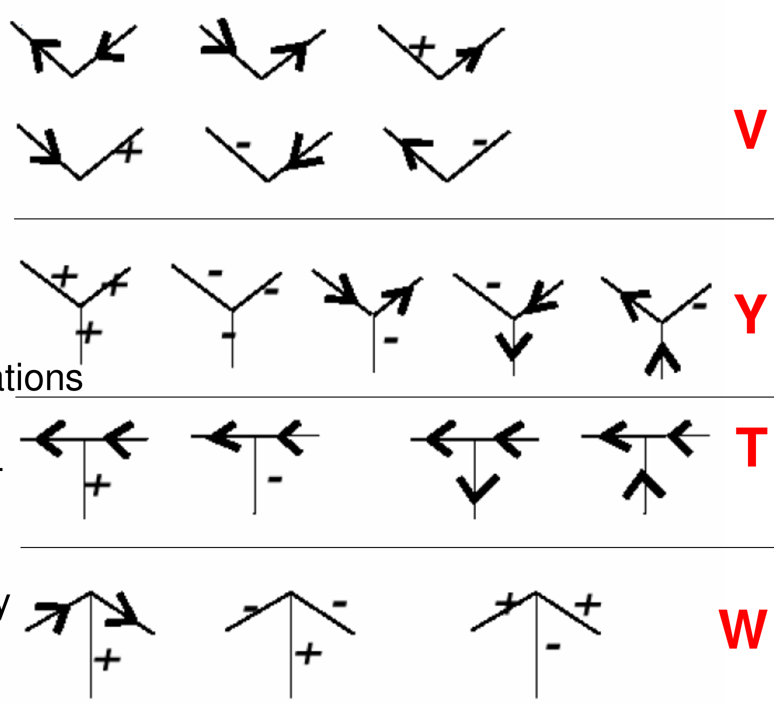
CSP Formulation

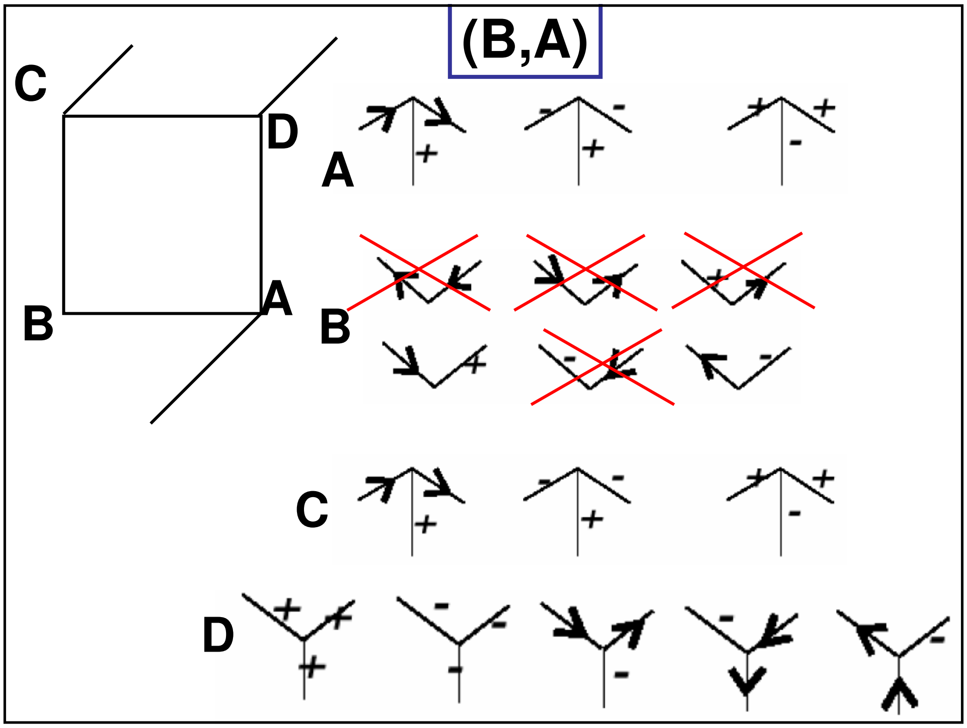
- Domain *D* = dictionary of 18 junction configurations
- Constraints: The line joining two junctions must have single label in {-,+, →}
- Problem: Assign values to all the junctions such that all of the edges are labeled
- Solved by constraint propagation: *Waltz labeling algorithm*

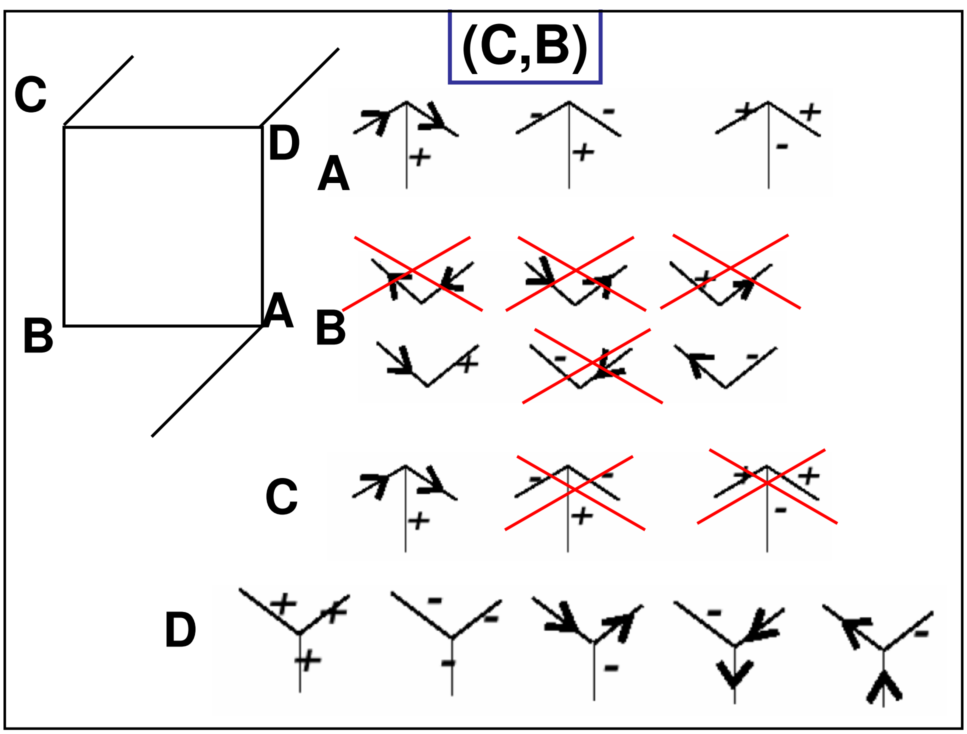


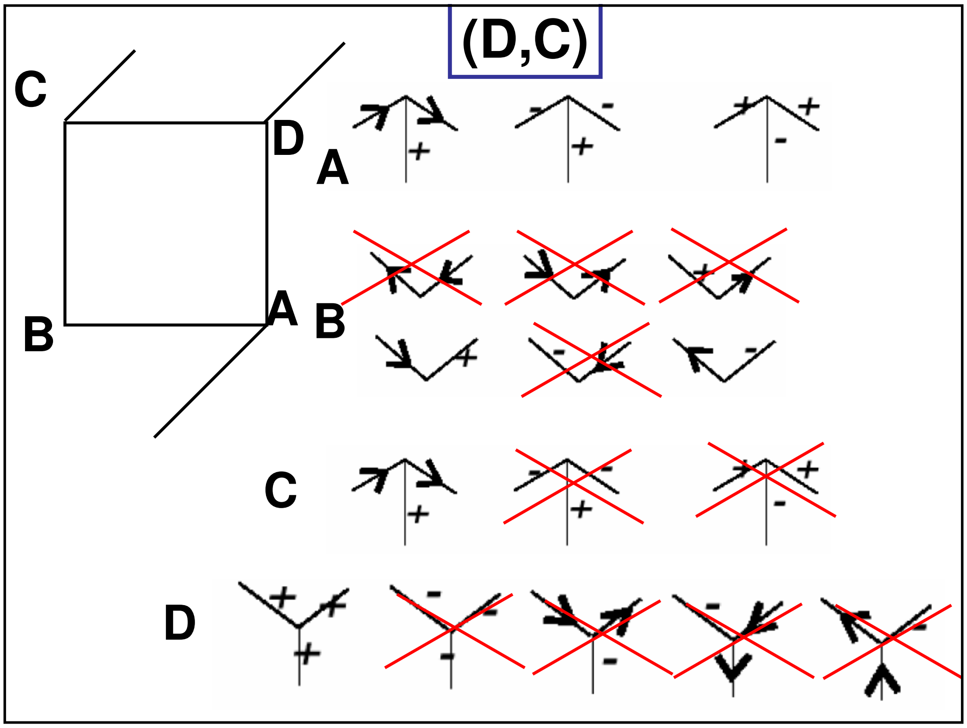
Only 18 possible junction configurations

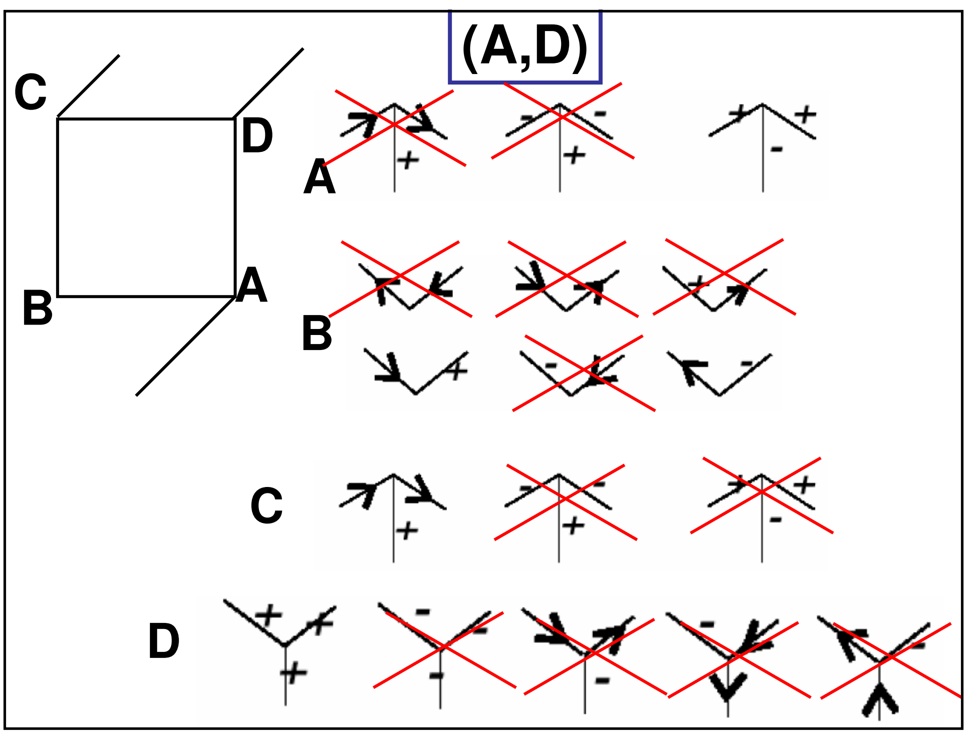
Huffman-Clowes junction dictionary

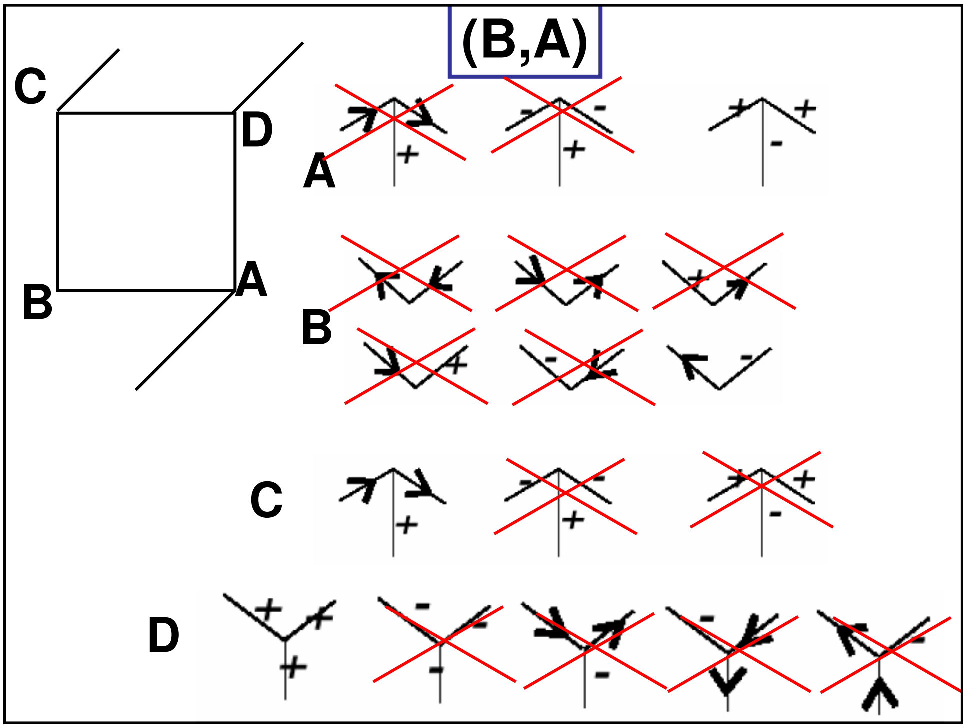


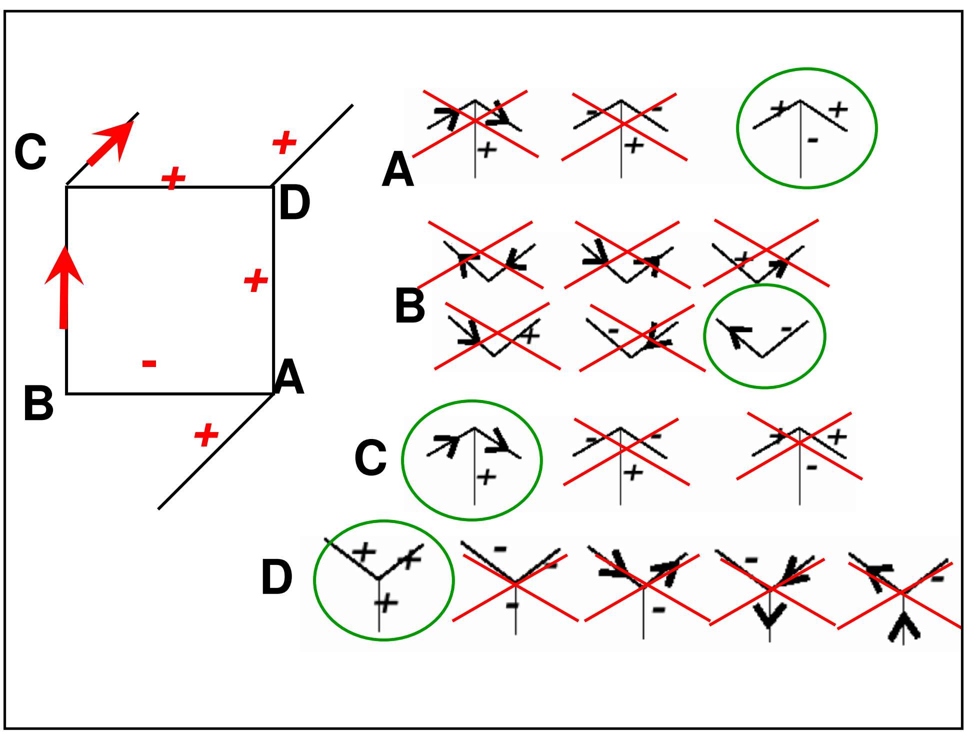

(B,A)
C
D
A
B
A
B
C
D


(C,B)
C
D
A
B
A
B
C
D


(D,C)
C
D
A
B
A
B
C
D
+
-


(A,D)
C
D
A
B
A
B
C
D
+
-


(B,A)
C
D
A
B
A
B
C
D


C
D
A
B
A
B
C
D

Labeling Notes

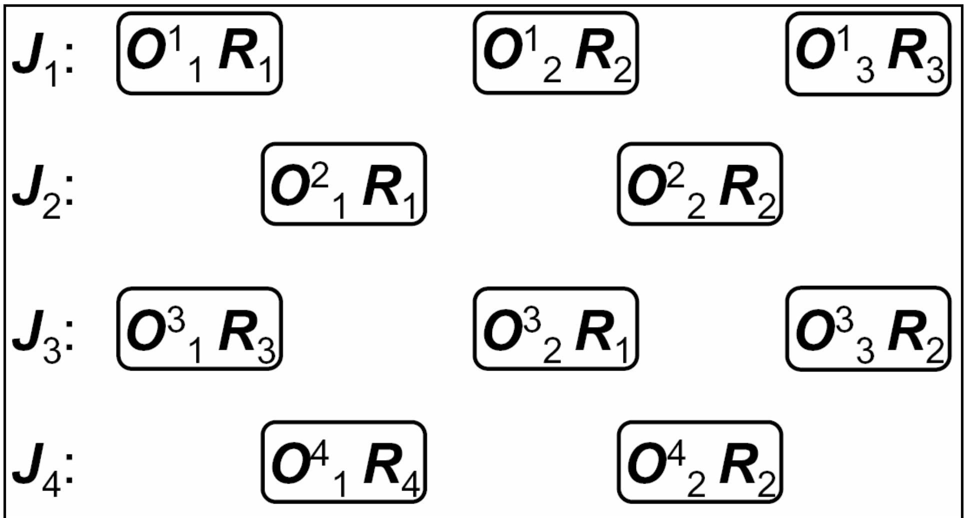
- Extended to include shadows and tangent contact (10 junction types and a much larger number of valid configurations)
- Key observation: ***Computation grows (roughly) linearly with the number of edges!***

CP for line labeling described in detail in P. Winston, "Artificial Intelligence", MIT Press

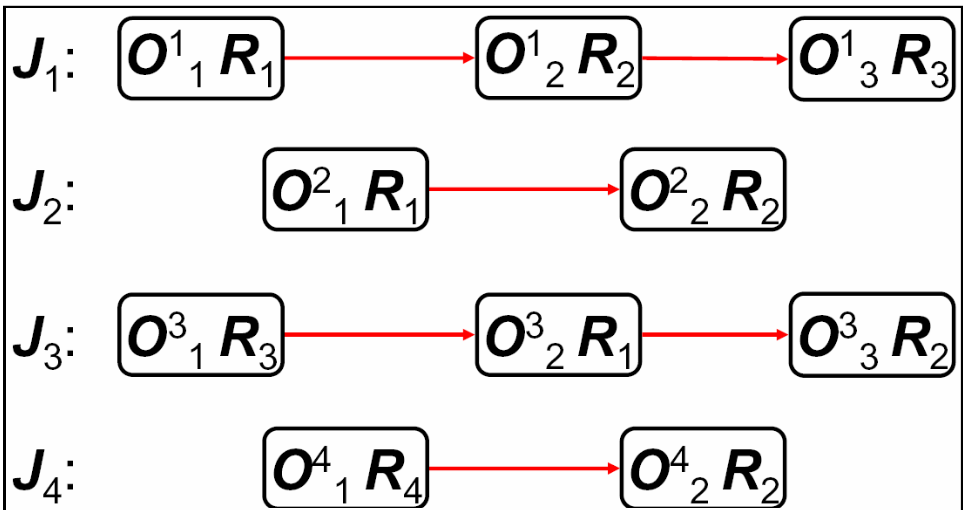
Example: Scheduling

- A set of *N* Jobs $\{\boldsymbol{J}_1,..,\boldsymbol{J}_N\}$ needs to be completed
- Each job *j* is composed of a set of L_j operations $\{\boldsymbol{O}^j_1,..,\boldsymbol{O}^j_{Lj}\}$ to be executed sequentially
- Each task $\boldsymbol{O}^j_i$ has a known duration T^j_i
- Tasks may need to use resources out of a pool of *M* resources $\{\boldsymbol{R}_1,..,\boldsymbol{R}_M\}$
- A resource cannot be used by two operations at the same time
- All jobs must be completed by time $t = Due$
- Problem: Schedule the start time of each operation S^j_i using discrete times $\{0,\ldots,T\}$

See recent survey in www.cs.cmu.edu/afs/cs/user/sfs/www/mista03/mista03.html
Illustrations from N. Sadeh and M.S. Fox. "Variable and Value Ordering Heuristics for the Job Shop Constraint Satisfaction Problem"



- 4 jobs
- 4 resources
- 10 operations



Precedence constraints:

$S^1_1 + T^1_1 <= S^1_2$

$S^1_2 + T^1_2 <= S^1_3$

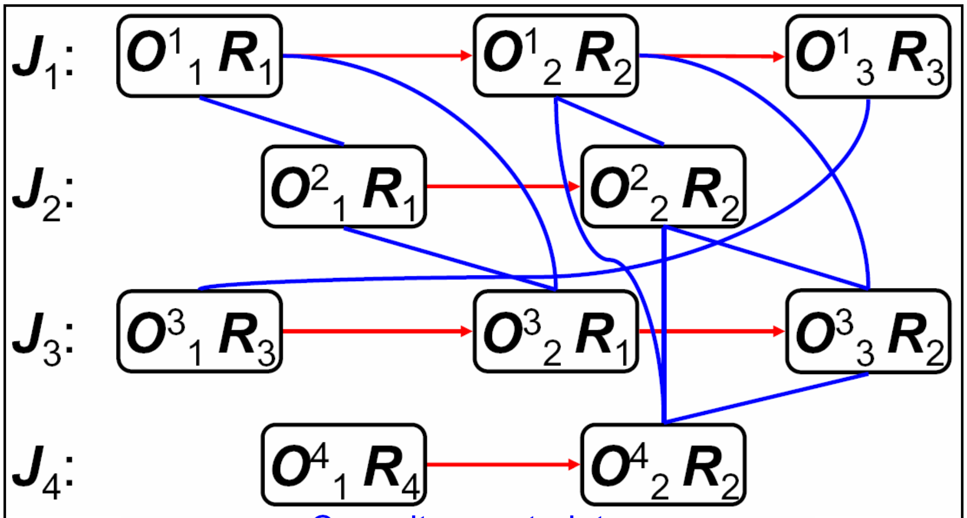
....

Delivery constraints:

$S^1_3 + T^1_3 <= Due$

$S^2_2 + T^2_2 <= Due$

$S^3_3 + T^3_3 <= Due$



Capacity constraints:

$S^1_1 + T^1_1 <= S^3_2$ OR $S^3_2 + T^3_2 <= S^1_1$

(Operations (1,1) and (3,2) share the same resource (R1) so either (1,1) is fully completed before (1,2) or (1,2) is completed before (1,1)

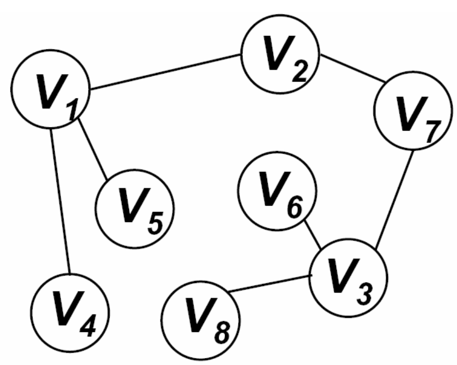
Generic CSP Solution

- Repeat until all variables have been assigned:
- Apply a consistency enforcement procedure
  - Forward checking
  - Constraint propagation
- If no solutions left:
  - Backtrack to a previous variable
- Else
  - select the next variable to be assigned
    - Using variable ordering heuristic
  - Select a value to try for this variable
    - Using value ordering heuristic

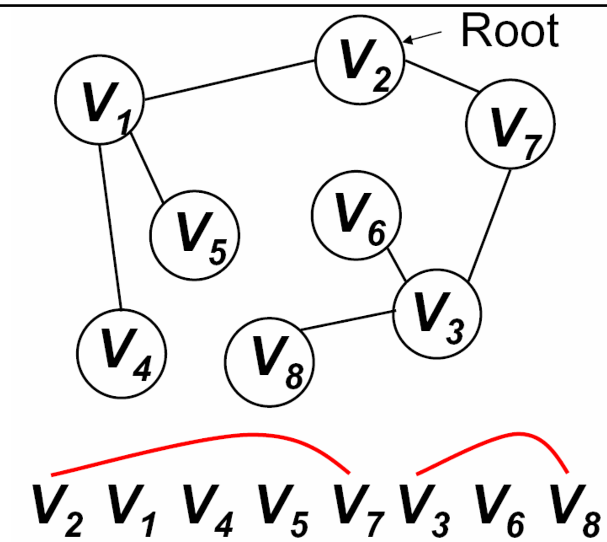
Outline

- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems

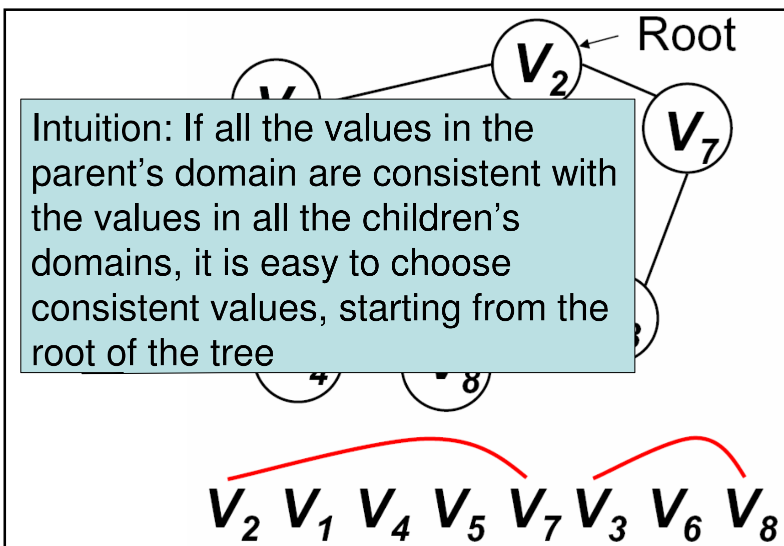
Important Special Case: Constraint Trees



- Constraint graph is a tree: Two variables are connected by one path
- Can always be solved in *linear* time in the number of variables



Order the variables such that the parent of a node appears always before that node in the list



Intuition: If all the values in the parent's domain are consistent with the values in all the children's domains, it is easy to choose consistent values, starting from the root of the tree

Order the variables such that the parent of a node appears always before that node in the list

Constraint Tree Algorithm

1. Up from leaves to root:
   - For every variable V_i, starting at the leaves:
   - $V_j = parent(V_i)$
   - Remove all the values x in $D(V_j)$ for which there is no consistent value in $D(V_i)$
2. Down from root to leaves:
   - Assign a value to the root of the tree
   - For every variable V_i:
     - Choose a value x in $D(V_i)$ consistent with the value assigned to $parent(V_i)$

Constraint Tree Algorithm

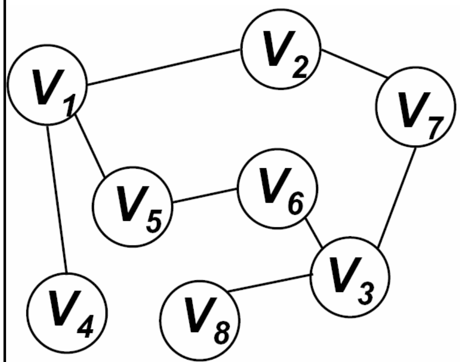
1. Up from leaves to root:
   - For every variable V_i, starting at the leaves:
   - $V_j = parent(V_i)$
   - Remove all the values x in $D(V_j)$ for which there is no consistent value in $D(V_i)$
2. Down from root to leaves:
   - Assign a value to the root of the tree
   - For every variable V_i:
     - Choose a value x in $D(V_i)$ consistent with the value assigned to $parent(V_i)$

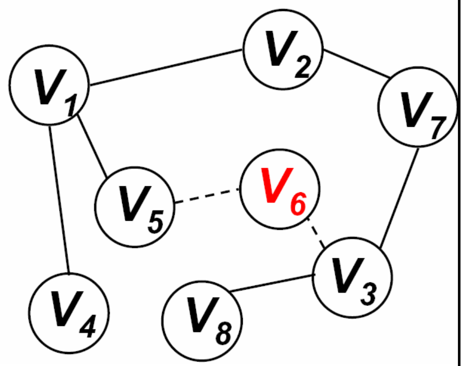
Visit each variable once: N

Worst case: Need to check all pais of values: d^2

Total time: $O(N d^2)$

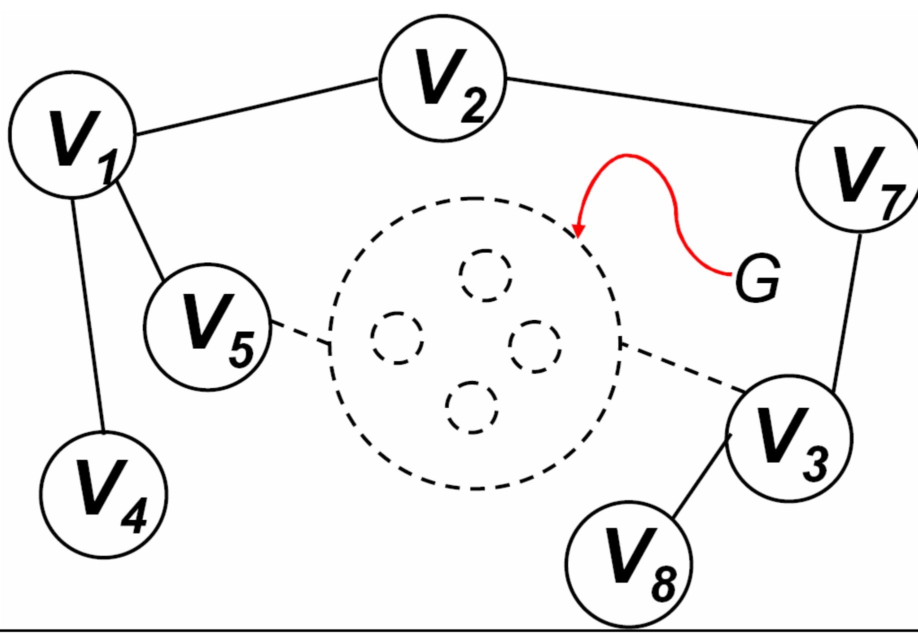
Almost Tree

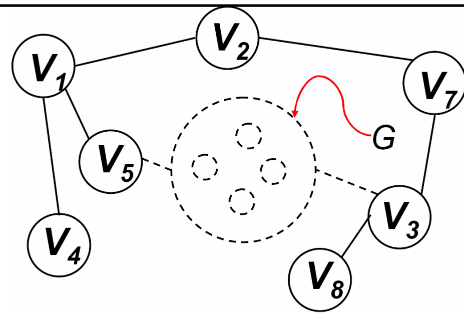




- The constraint graph becomes a tree once a value is chosen for V_6
- We don't know which value to choose → Try all possible values

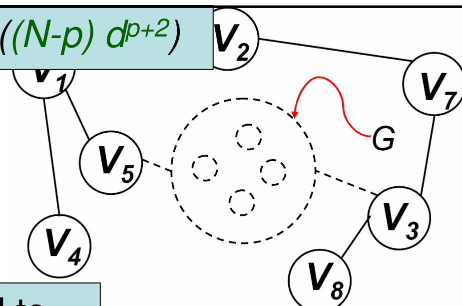
More General Case





- Removing a connected group *G* of *p* variables transforms the graph into a tree problem that can be solved efficiently.
- We don't know how to set the variables in *G*:
  - For every possible consistent assignment of values to variables in *G*:
    - Apply the tree algorithm to the rest of the variables

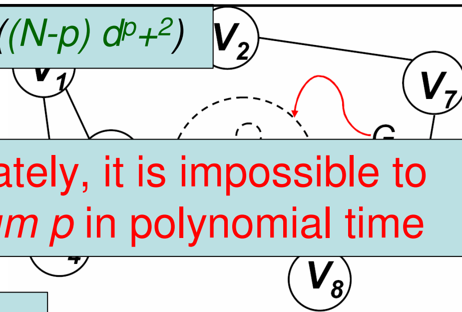
Complexity: $O((N-p)\, d^{p+2})$



Worst case: Need to check all possible assignments in *G* → d^p

Tree algorithm → $(N-p)\, d^2$

- Removing a connected group *G* of *p* variables transforms the graph into a tree problem that can be solved efficiently.
- We don't know how to set the variables in *G*:
  - For every possible consistent assignment of values to variables in *G*:
    - Apply the tree algorithm to the rest of the variables



Complexity: O((N-p) d^{p+2})

Note: Unfortunately, it is impossible to find the *minimum p* in polynomial time

Worst case: Need to check all possible assignments in *G* → d^p

…nected group *G* of *p* …rms the graph into a tree problem … be solved efficiently.

- We don't …ow h… *G*:

Tree algorithm → *(N-p)* d^2

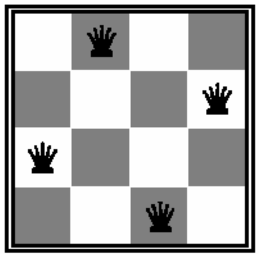
  - For every possible consistent assignment of values to variables in *G*:
    - Apply the tree algorithm to the rest of the variables

Outline

- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems

Local Search Techniques for CSP

N-Queens



SAT

$A \vee \neg B \vee C$

$\neg A \vee C \vee D$

$B \vee D \vee \neg E$

$\neg C \vee \neg D \vee \neg E$

$\neg A \vee \neg C \vee E$

- These problems can be formulated as CSPs
- We have used local search methods to solve them in an earlier lecture (hill climbing, annealing, tabu search, genetic algorithms)
- When are local search methods applicable?
  - Direct solution through local search effective for some problems
  - Optimization of a cost function in addition to CSP
  - Online update of CSP solution

Local Search for CSP

State = assignment of values to all the variables

V_1	V_2	V_3	V_4	V_5	V_6
a	*b*	*c*	*d*	*e*	*f*

Move = Change one variable

V_1	V_2	V_3	V_4	V_5	V_6
a	*b*	*c'*	*d*	*e*	*f*

Evaluation = number of conflicts (non-satisfied constraints) between variables

Generic Local Search: Min-Conflicts Algorithm

- Start with a complete assignment of variables
- Repeat until a solution is found or maximum number of iterations is reached:
  - Select a variable V_i *randomly* among the variables *in conflict*
  - Set V_i to the value that *minimizes* the number of constraints violated

---

- Far more effective than CSP search for many problems
- All previous variants of hill-climbing are applicable
- Generic form similar to WALKSAT seen earlier

- Start ... varia...
- Repe... maxi... ...ched:
  - Select a variable V_i *randomly* among the variables *in conflict*
  - Set V_i to the value that *minimizes* the number of constraints violated

	USA	N-Queens (1<N<=50)	Zebra
DFS Backtracking	$> 10^6$	$> 40\ 10^6$	$3.9\ 10^6$
+ MRV	$> 10^6$	$13.5\ 10^6$	1,000
Forward Checking	2,000	$> 40\ 10^6$	35,000
+ MRV	60	817,000	500
Min-Conflicts	64	4,000	2,000

(Data from Russell & Norvig)

MRV heuristic is always very effective

Local search is surprisingly effective.
Can solve N-queens efficiently for $N = 10^7$!!
Why are such problems “easy” to solve??

	USA	N-Queens (1<N<=50)	Zebra
DFS Backtracking	[illegible]	[illegible]	[illegible]
+ MRV	[illegible]	$13.5\ 10^6$	1,000
Forward Checking	[illegible]	$> 40\ 10^6$	35,000
+ MRV	60	817,000	500
Min-Conflicts	64	4,000	2,000

Outline

- Definitions
- Standard search
- Improvements
  - Backtracking
  - Forward checking
  - Constraint propagation
- Heuristics:
  - Variable ordering
  - Value ordering
- Examples
- Tree-structured CSP
- Local search for CSP problems