CS 1571 Introduction to AI
Lecture 5

Informed (heuristic) search (cont). Constraint-satisfaction search.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Administration

- **PS–1 due today**
  - Report before the class begins
  - Programs through ftp
- **PS-2 is out**
  - on the course web page
  - due next week on Tuesday, September 17, 2002
    - Report
    - Programs

Evaluation-function driven search

- A search strategy can be defined in terms of **a node evaluation function**
- **Evaluation function**
  - Denoted $f(n)$
  - Defines the desirability of a node to be expanded next
- **Evaluation-function driven search: expand the node (state) with the best evaluation-function value**
- **Implementation:** successors of the expanded node are inserted into the **priority queue** in the decreasing order of their evaluation function value

Uniform cost search

- **Uniform cost search (Dijkstra's shortest path):**
  - A special case of the evaluation-function driven search

$$f(n) = g(n)$$

- **Path cost function** $g(n)$;
  - path cost from the initial state to n
- **Uniform-cost search:**
  - Can handle general minimum cost path-search problem:
  - **weights or costs** associated with operators (links).
- **Note:** Uniform cost search relies on the problem definition only
  - Uninformed search method

Best-first search

Best-first search

- incorporates a **heuristic function,** $h(n)$, into the evaluation function $f(n)$.
- **heuristic function:** measures a potential of a state (node) to reach a goal

Special cases (differ in the design of evaluation function):

- **Greedy search**

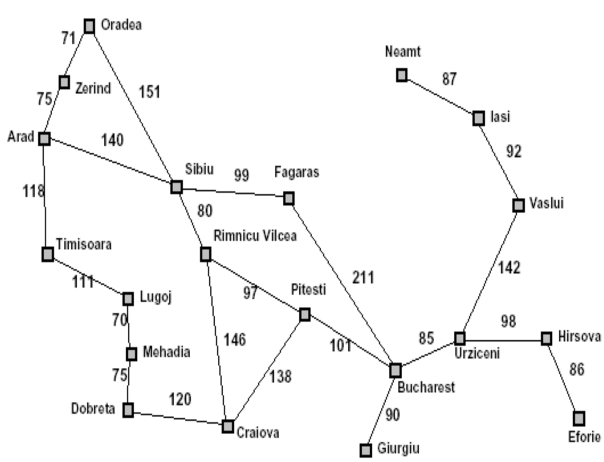
$$f(n) = h(n)$$

- **A* algorithm**

$$f(n) = g(n) + h(n)$$

+ **iterative deepening** version of A* : **IDA***

Example: traveler problem with straight-line distance information



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Straight-line distances** give an estimate of the cost of the path between the two cities

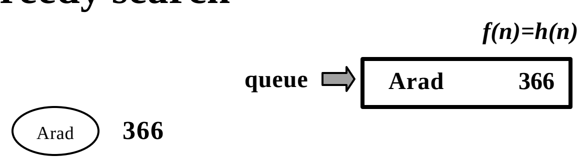
Greedy search method

- Evaluation function is equal to the heuristic function

$$f(n) = h(n)$$

- **Idea:** the node that seems to be the closest to the goal is expanded first

Greedy search

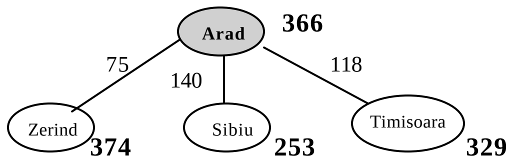


Greedy search

f(n)=h(n)

queue

Sibiu	**253**
Timisoara	**329**
Zerind	**374**

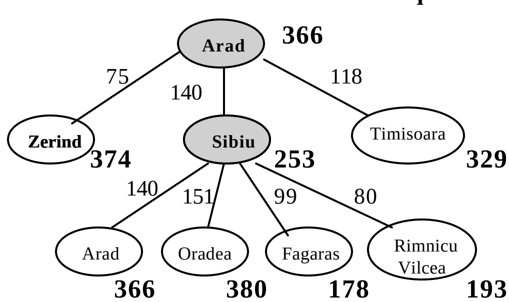


Greedy search

f(n)=h(n)

queue

Fagaras	**99**
Rimniciu V.	**193**
Timisoara	329
Arad	**366**
Zerind	374
Oradea	**380**

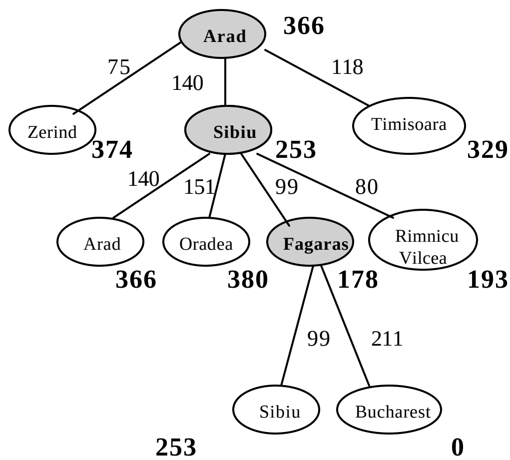


Greedy search

f(n)=h(n)

queue

Bucharest	**0**
Rimniciu V.	193
Sibiu	**253**
Timisoara	329
Arad	366
Zerind	374
Oradea	380



Properties of greedy search

- **Completeness:** ?
- **Optimality: ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**

Properties of greedy search

- **Completeness: No**.
  We can loop forever. Nodes that seem to be the best choices can lead to cycles. Elimination of state repeats can solve the problem.
- **Optimality: No.**
  Even if we reach the goal, we may be biased by a bad heuristic estimate. Evaluation function disregards the cost of the path built so far.
- **Time complexity:** $O(b^m)$
  Worst case !!! But often better!
- **Memory (space) complexity:** $O(b^m)$
  Often better!

A* search

- The problem with the greedy search is that it can keep expanding paths that are already very expensive.
- The problem with the uniform-cost search is that it uses only past exploration information (path cost), no additional information is utilized
- **A* search**

$$f(n) = g(n) + h(n)$$

$g(n)$ - cost of reaching the state

$h(n)$ - estimate of the cost from the current state to a goal

$f(n)$ - estimate of the path length

- **Additional A*condition**: admissible heuristic

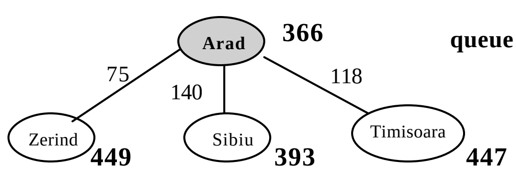
$$h(n) \leq h^*(n) \quad \text{for all } n$$

A* search example

queue ➡

	f(n)
Arad	**366**

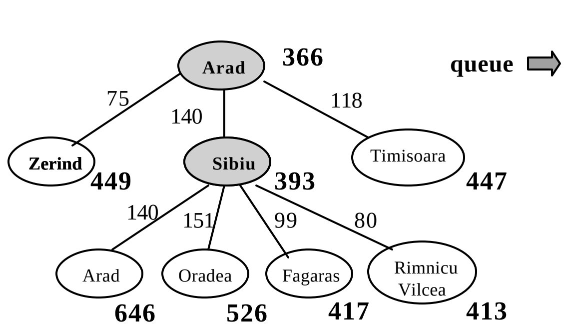
A* search example



queue ➡

	f(n)
Sibiu	**393**
Timisoara	**447**
Zerind	**449**

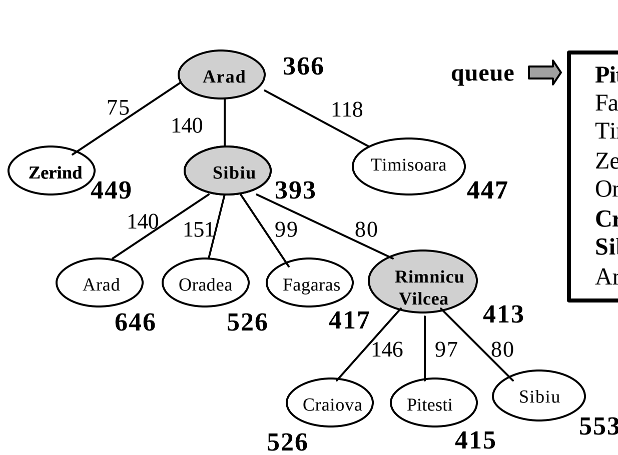
A* search example



Rimnicu V.	**413**
Fagaras	**417**
Timisoara	447
Zerind	449
Oradea	**526**
Arad	**646**

CS 1571 Intro to AI

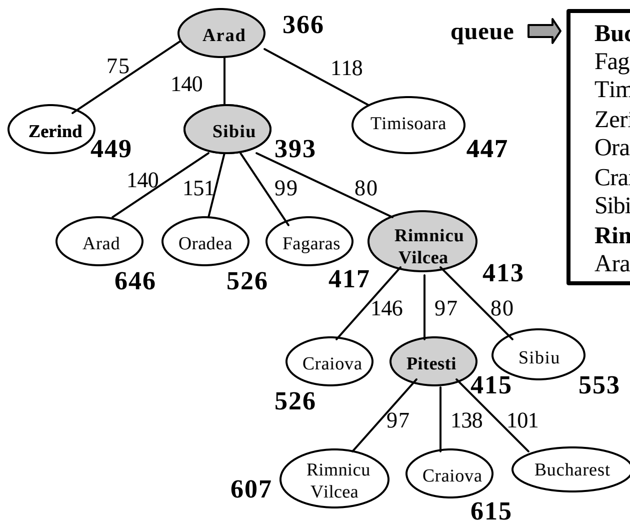
A* search example



Pitesti	**415**
Fagaras	417
Timisoara	447
Zerind	449
Oradea	526
Craiova	**526**
Sibiu	**553**
Arad	646

CS 1571 Intro to AI

A* search example



f(n)

Bucharest	**0**
Fagaras	417
Timisoara	447
Zerind	449
Oradea	526
Craiova	526
Sibiu	553
Rimnicu V.	**607**
Arad	646

Properties of A* search

- **Completeness: ?**
- **Optimality: ?**
- **Time complexity:**
  - **?**
- **Memory (space) complexity:**
  - **?**

Properties of A* search

- **Completeness: Yes.**
- **Optimality: Yes (with the admissible heuristic)**
- **Time complexity:**
  - **Order roughly the number of nodes with *f(n)* smaller than the cost of the optimal path *g****
- **Memory (space) complexity:**
  - **Same as time complexity (all nodes in the memory)**

Optimality of A*

- In general, a heuristic function $h(n)$:

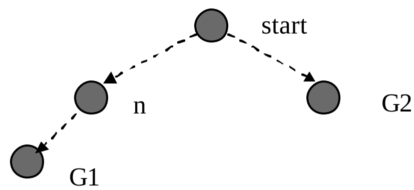
  Can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$
- Is the A* optimal for the arbitrary heuristic function?
- **No !**
- **Admissible heuristic condition**
  - **Never overestimate the distance to the goal !!!**

    $h(n) \leq h^*(n)$ for all n

    Example: the straight-line distance in the travel problem never overestimates the actual distance
- **Claim: A* search (with admissible heuristic !!) is optimal**

Optimality of A* (proof)

- Let G1 be the optimal goal (with the minimum path distance). Assume that we have a sub-optimal goal G2. Let *n* be a node that is on the optimal path and is in the queue together with G2



Then:

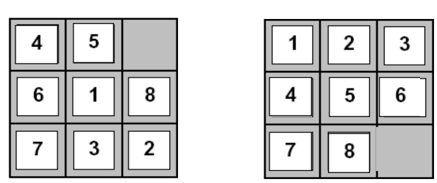
$$
\begin{aligned}
f(G2) &= g(G2) && \text{since } h(G2)=0 \\
&> g(G1) && \text{since G2 is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

And thus **A* never selects G2 before *n***

Admissible heuristics

- Heuristics are designed based on relaxed version of problems
- **Example:** the 8-puzzle problem

Initial position **Goal position**



- Admissible heuristics:
  1. number of misplaced tiles
  2. Sum of distances of all tiles from their goal positions (Manhattan distance)

Admissible heuristics

- We can have multiple admissible heuristics for the same problem
- **Dominance:** Heuristic function h_1 dominates h_2 if

$$\forall n \quad h_1(n) \geq h_2(n)$$

- **Combination:** two or more admissible heuristics can be combined to give a new admissible heuristics
  - Assume two admissible heuristics h_1, h_2

Then: $h_3(n) = \max(h_1(n), h_2(n))$

is admissible

IDA*

Iterative deepening version of A*

- Progressively increases the evaluation function limit (instead of the depth limit)
- Performs limited-cost depth-first search for the current evaluation function limit
  - Keeps expanding nodes in the depth first manner up to the evaluation function limit

Problem: the amount by which the evaluation limit should be progressively increased

Solutions:

- **peak over the previous step boundary** to guarantee that in the next cycle more nodes are expanded
- **Increase the limit by the fixed cost increment – say u**

IDA*

Solution 1: peak over the previous step boundary to guarantee that in the next cycle more nodes are expanded

Properties:

- the choice of the new cost limit influences how many nodes are expanded in each iteration
- We may find the sub-optimal solution
  - **Fix:** complete the search up to the limit to find the best

Solution 2: Increase the limit by a fixed cost increment (u)

Properties:

- Too many or too few nodes expanded – no control of the number of nodes
- The solution of accuracy < u is found

Constraint satisfaction search

Search problem

A search problem:

- **Search space (or state space):** a set of objects among which we conduct the search;
- **Initial state:** an object we start to search from;
- **Operators (actions):** transform one state in the search space to the other;
- **Goal condition:** describes the object we search for

- **Possible metric on a search space:**
  - measures the quality of the object with regard to the goal

Search problems occur in planning, optimizations, learning

Constraint satisfaction problem (CSP)

Two types of search:

- **path search** (a path from the initial state to a state satisfying the goal condition)
- **configuration search (**a configuration satisfying goal conditions)

Constraint satisfaction problem (CSP) is **a configuration search problem** where**:**

- A state is defined by a set of variables
- Goal condition is represented by a set constraints on possible variable values

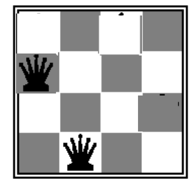
Special properties of the CSP allow more specific procedures to be designed and applied for solving them

Example of a CSP: N-queens

Goal: n queens placed in non-attacking positions on the board

Variables:

- Represent queens, one for each column:
  - Q_1, Q_2, Q_3, Q_4
- Values:
  - Row placement of each queen on the board {1, 2, 3, 4}



$Q_1 = 2, Q_2 = 4$

Constraints: $Q_i \neq Q_j$ Two queens not in the same row

$| Q_i - Q_j | \neq | i - j |$ Two queens not on the same diagonal

Satisfiability (SAT) problem

Determine whether a sentence in the conjunctive normal form (CNF) is satisfiable (can evaluate to true)
- Used in the propositional logic (covered later)

$$(P \vee Q \vee \neg R) \wedge (\neg P \vee \neg R \vee S) \wedge (\neg P \vee Q \vee \neg T) \ldots$$

Variables:

- Propositional symbols (P, R, T, S)
- Values: *True, False*

Constraints:

- Every conjunct must evaluate to true, at least one of the literals must evaluate to true

$$(P \vee Q \vee \neg R) \equiv True, (\neg P \vee \neg R \vee S) \equiv True, \ldots$$

Other real world CSP problems

Scheduling problems:

- E.g. telescope scheduling
- High-school class schedule

Design problems:

- Hardware configurations
- VLSI design

More complex problems may involve:

- **real-valued variables**
- **additional preferences on variable assignments** – the optimal configuration is sought

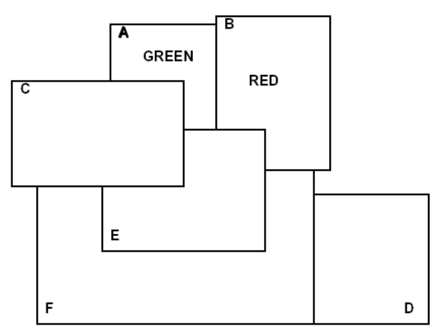
Map coloring

Color a map using k different colors such that no adjacent countries have the same color

Variables:

- Represent countries
  - A, B, C, D, E
- Values:
  - K -different colors

  {Red, Blue, Green,..}



Constraints: $A \neq B, A \neq C, C \neq E$, etc

An example of a problem with **binary constraints**

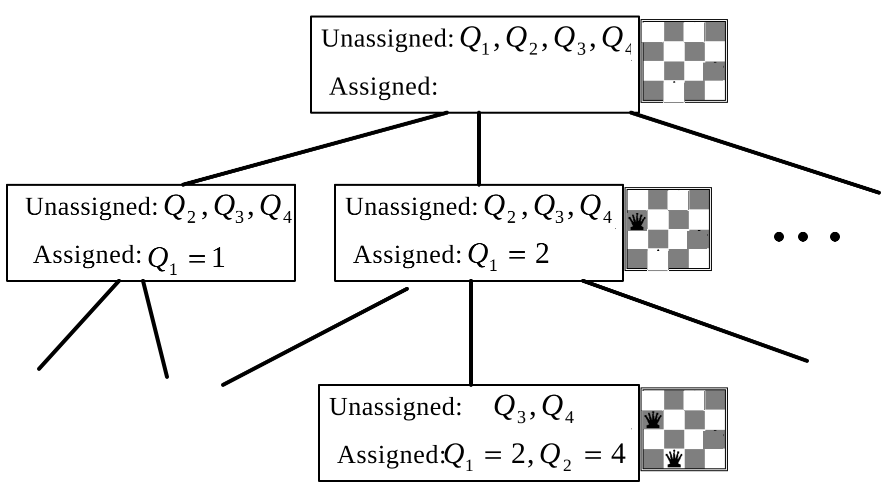
Constraint satisfaction as a search problem

Formulation of a CSP as a search problem:

- **States.** Assignment (partial, complete) of values to variables.
- **Initial state.** No variable is assigned a value.
- **Operators.** Assign a value to one of the unassigned variables.
- **Goal condition.** All variables are assigned, no constraints are violated.

- **Constraints** can be **represented**:
  - **Explicitly** by a set of allowable values
  - **Implicitly** by a function that tests for the satisfaction of constraints

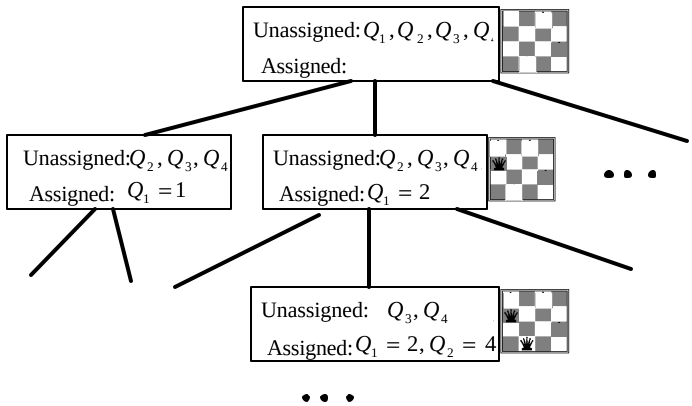
Solving CSP as a standard search



CS 1571 Intro to AI

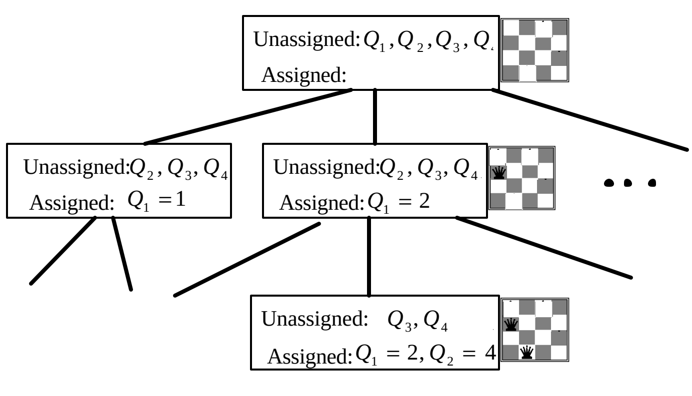
Solving a CSP through standard search

- **Maximum depth of the tree (m): ?**
- **Depth of the solution (d) : ?**
- **Branching factor (b) : ?**



Solving a CSP through standard search

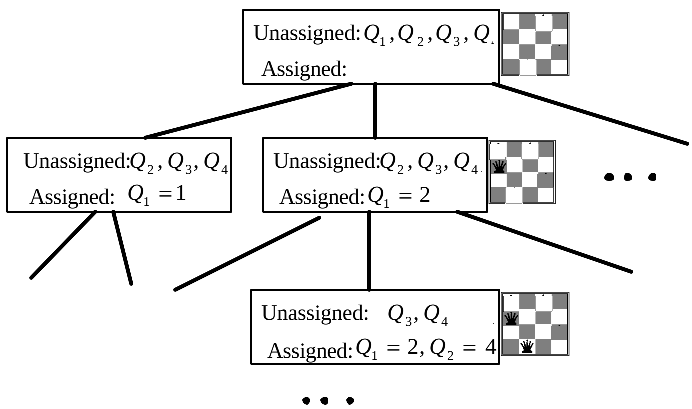
- **Maximum depth of the tree:** Number of variables of the CSP
- **Depth of the solution:** Number of variables of the CSP
- **Branching factor:** if we fix the order of variable assignments the branch factor depends on the number of their values



Solving a CSP through standard search

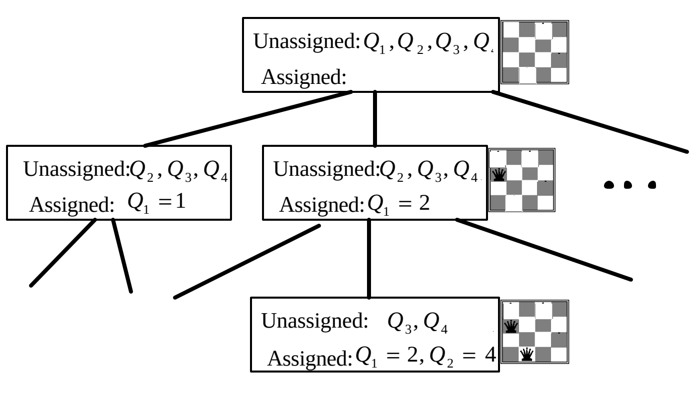
- **What search algorithm to use: ?**

  Depth of the tree = Depth of the solution=number of vars



Solving a CSP through standard search

- **What search algorithm to use: Depth first search !!!**
  - Since we know the depth of the solution
  - We do not have to keep large number of nodes in queues



Backtracking

Depth-first search for CSP is also referred to as **backtracking**

The violation of constraints needs to be checked for each node, either during its generation or before its expansion

Important problem:

- Current variable assignments in concert with constraints restrict remaining legal values of unassigned variables;
- The remaining legal and illegal values of variables may be inferred (effect of constraints propagates)
- It is necessary to keep track of the remaining legal values, so we know when the constraints are violated and when to terminate the search

Constraint propagation

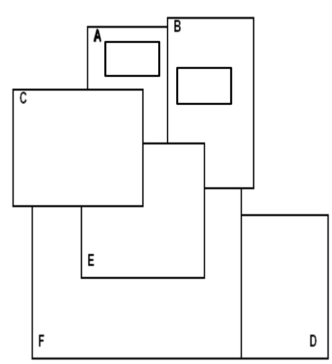
A **state** (more broadly) is defined by a set variables and their legal and illegal assignments

Legal and illegal assignments can be represented through variable **equations** and variable **disequations**

Example: **map coloring**

Equation $A = \text{Red}$

Disequation $C \neq \text{Red}$



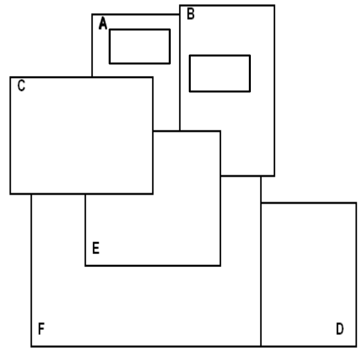
Constraints + assignments can entail new equations and disequations

$A = \text{Red} \rightarrow B \neq \text{Red}$

Constraint propagation

- Assign A=Red

	Red	Blue	Green
A	✓		
B			
C			
D			
E			
F			

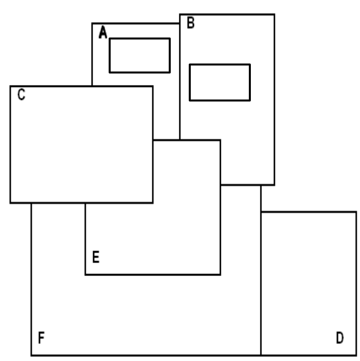


✓ - equations ✗ - disequations

Constraint propagation

- Assign A=Red

	Red	Blue	Green
A	✓		
B	✗		
C	✗		
D			
E	✗		
F			

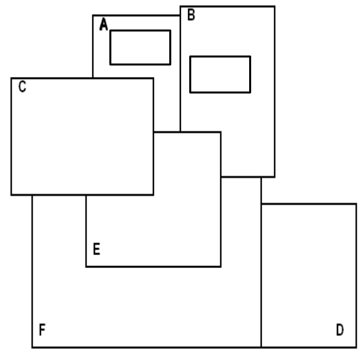


✓ - equations ✗ - disequations

Constraint propagation

- Assign E=Blue

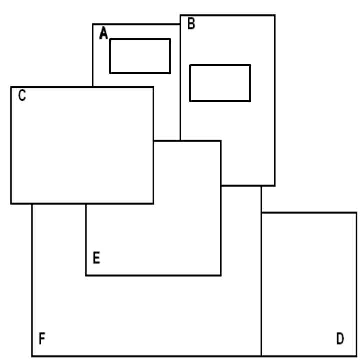
	Red	Blue	Green
A	✓		
B	✗		
C	✗		
D			
E	✗	✓	
F			



Constraint propagation

- Assign E=Blue

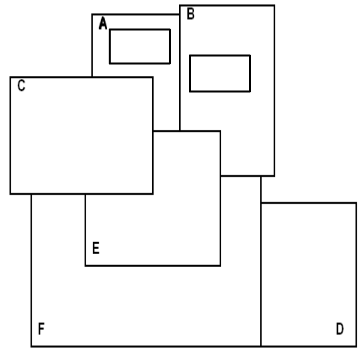
	Red	Blue	Green
A	✓	✗	
B	✗	✗	
C	✗	✗	
D			
E	✗	✓	
F		✗	



Constraint propagation

- Assign F=Green

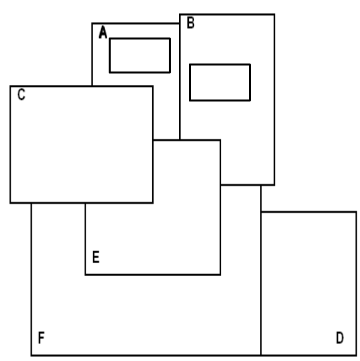
	Red	Blue	Green
A	✓	✗	
B	✗	✗	
C	✗	✗	
D			
E	✗	✓	
F		✗	✓



Constraint propagation

- Assign F=Green

	Red	Blue	Green
A	✓	✗	
B	✗	✗	✗
C	✗	✗	✗
D			✗
E	✗	✓	✗
F		✗	✓

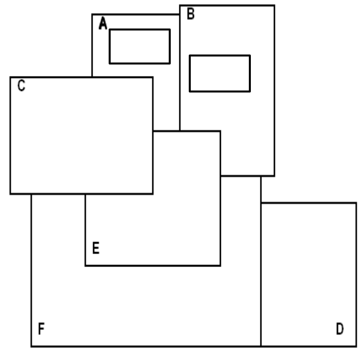


Conflict !!! No legal assignments available for B and C

Constraint propagation

- We can derive remaining legal values through propagation

	Red	Blue	Green
A	✓	✗	
B	✗	✗	✓
C	✗	✗	✓
D			
E	✗	✓	
F		✗	



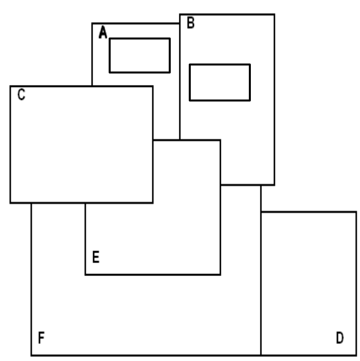
B=Green

C=Green

Constraint propagation

- We can derive remaining legal values through propagation

	Red	Blue	Green
A	✓	✗	✗
B	✗	✗	✓
C	✗	✗	✓
D	✗		
E	✗	✓	✗
F	✓	✗	✗



B=Green

C=Green

F=Red

Constraint propagation

Three known techniques for propagating the effects of past assignments and constraints:

- **Value propagation**
- **Arc consistency**
- **Forward checking**

- **Difference:**
  - Completeness of inferences
  - Time complexity of inferences.