CS 188: Artificial Intelligence Spring 2009

Lecture 4: Constraint Satisfaction
2/3/2009

John DeNero – UC Berkeley
Slides adapted from Dan Klein, Stuart Russell or Andrew Moore

Announcements

- Project 1 (Search) is due tomorrow
  - Come to office hours if you're stuck
    - Today at 1pm (Nick) and 3pm (John)
    - Tomorrow at 11am (John)
  - Up to 2 slip days (Friday at 11:59pm)
  - Issues: nodes expanded and hashing lists
- Written assignment 1 is due next Tuesday
  - Work in groups, write up alone
  - Printed copies are available after lecture today
  - Due at the beginning of Tuesday lecture

Today

- Backtracking Search Recap
- Structure in CSPs
- Local Search Algorithms

Production Scheduling

	F430 Spyder	599 GTB	612 Scaglietti
			
Convertible	+		
V12		+	+

Can't schedule two convertibles or two V12 engines in a row

Example Adapted from Edward Tsang

Production Scheduling

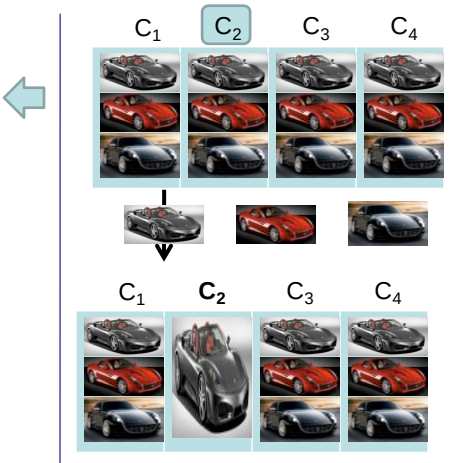
	F(4)30 Spyder	(5)99 GTB	(6)12 Scaglietti
			
Convertible	+		
V12		+	+

- Variables: C_1, C_2, C_3, …
- Values: 4, 5, 6
- Constraints:
  - (C_i , C_{i+1}) in { (4,5), (5,4), (4,6),(6,4) }

Backtracking Search Review

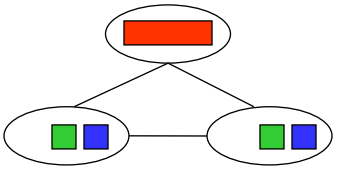
Backtracking:

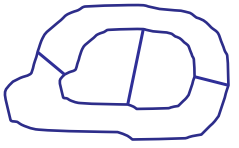
- Pick a variable (MRV, degree)
- Order values (LCV)
- For each value:
  - Instantiate variable
  - Forward checking
  - Arc consistency
  - Backtracking

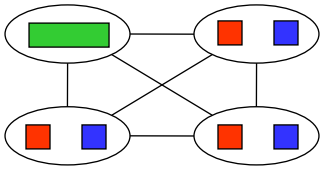


Limitations of Arc Consistency

- After running arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)

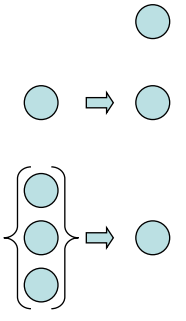


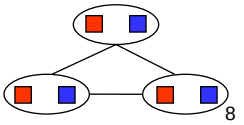




K-Consistency

- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the k^{th} node.
- Higher k more expensive to compute
- (You need to know the k=2 algorithm)





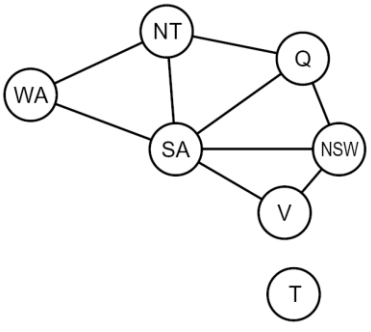
8

Strong K-Consistency

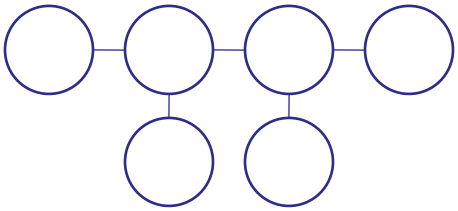
- Strong k-consistency: also k-1, k-2, … 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
  - Choose any assignment to any variable
  - Choose a new variable
  - By 2-consistency, there is a choice consistent with the first
  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - …
- Lots of middle ground between arc consistency and n-consistency!

Problem Structure

- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph
- Suppose each subproblem has c out of n total
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
  - E.g., n = 80, d = 2, c =20
  - 2^{80} = 4 billion years at 10 million nodes/sec
  - $(4)(2^{20})$ = 0.4 seconds at 10 million nodes/sec



Tree-Structured CSPs



- Tree structured mean no loops in the constraint graph
- Theorem: A tree-structured CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$
- Efficient algorithms for tree-structured problems also appear in probabilistic reasoning, where we have probability distributions over the values of each variable

11

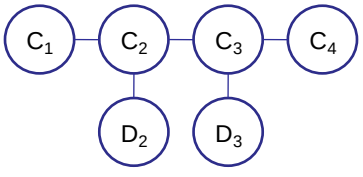
Example Tree-Structured CSP

	F(4)30 Spyder	(5)99 GTB	(6)12 Scaglietti
Convertible	+		
V12		+	+
Dino compatible	+		

Variables

C_i *in* { 4, 5, 6 }

D_i *in* {True, False}



Constraints

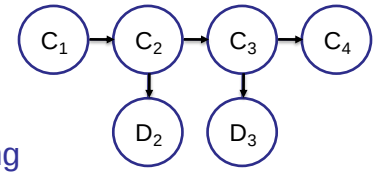
(C_i , C_{i+1}) *in*
{ (4,5), (5,4), (4,6), (6,4) }

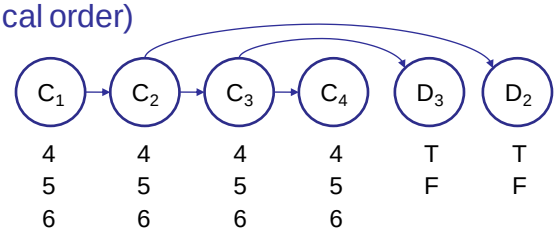
(C_i , D_i) *in*
{ (4,T), (4,F), (5,F), (6,F) }

D_3 = True

Solving a Tree-Structured CSP

- Choose any variable as root
- Order variables from root to leaves such that every node's parent precedes it in the ordering (topological order)



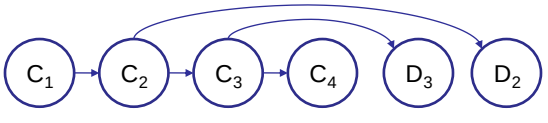


- Apply unary constraints
- For i = n : 2, apply Remove_Inconsistent(Parent(X_i), X_i)
- For i = 1 : n, assign X_i consistently with Parent(X_i)

13

Tree-Structured CSPs

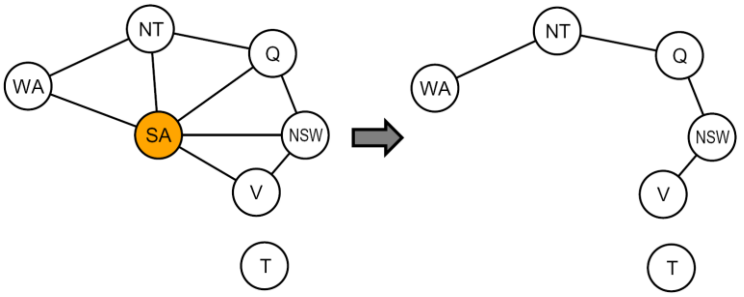
- Why does this work?
- Claim: After each node is processed leftward, all nodes to the right can be assigned in any way consistent with their parent.
- Proof: Induction on position



- Why doesn't this algorithm work with loops?
- Note: we'll see this basic idea again with Bayes' nets

14

Nearly Tree-Structured CSPs



- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime $O((d^c) (n-c) d^2)$, very fast for small c

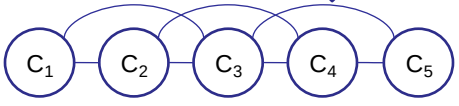
15

Tree Decompositions

- Create a tree-structured graph of subproblems
- Solve each subproblem to enforce local constraints
- Solve the CSP over subproblem mega-variables using our efficient tree-structured CSP algorithm

Constraint: Every convertible must be two steps after another

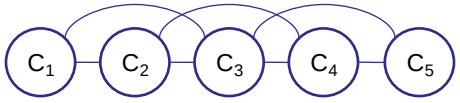
Constraint graph:

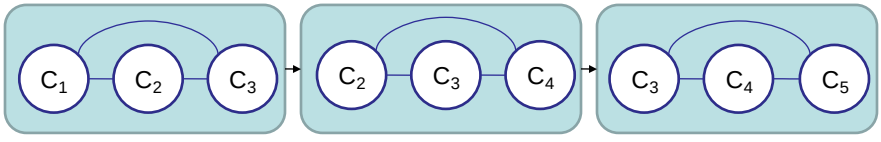


Tree Decompositions

- Create a tree-structured graph of subproblems
- Solve each subproblem to enforce local constraints
- Solve the CSP over subproblem mega-variables using our efficient tree-structured CSP algorithm

Constraint graph:





(4,5,4) (5,4,5) (6,4,5)
(C_1=4, C_2=5, C_3=4)
(4,6,4) (5,4,6) (6,4,6)

(4,5,4) (5,4,5) (6,4,5)
(C_2=5, C_3=4, C_4=6)
(4,6,4) (5,4,6) (6,4,6)

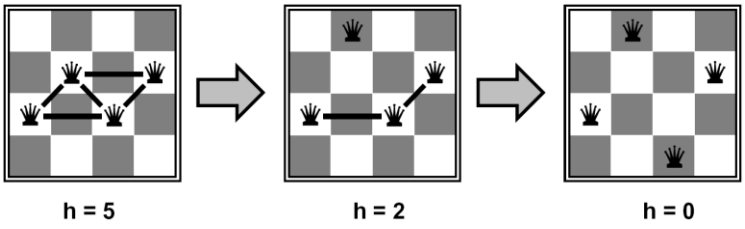
(4,5,4) (5,4,5) (6,4,5)
(C_3=4, C_4=6, C_5=4)
(4,6,4) (5,4,6) (6,4,6)

Iterative Algorithms for CSPs

- Local search methods typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - Start with some assignment with unsatisfied constraints
  - Operators *reassign* variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - Choose value that violates the fewest constraints
  - I.e., hill climb with h(n) = total number of violated constraints

18

Example: 4-Queens



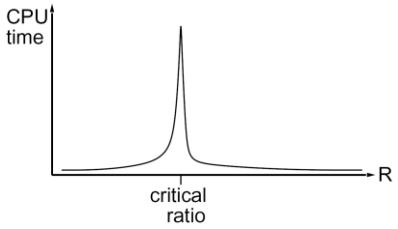
- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks

19

Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



20

Summary

- CSPs are a special kind of search 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 legal variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Constraint graphs allow for analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- Iterative min-conflicts is usually effective in practice

21

Local Search Methods

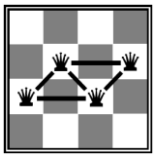
- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve what you have until you can't make it better
- Generally much faster and more memory efficient (but incomplete)

22

Types of Search Problems

- Planning problems:
  - We want a path to a solution (examples?)
  - Usually want an optimal path
  - *Incremental formulations*

- Identification problems:
  - We actually just want to know what the goal is (examples?)
  - Usually want an optimal goal
  - *Complete-state formulations*
  - Iterative improvement algorithms



23

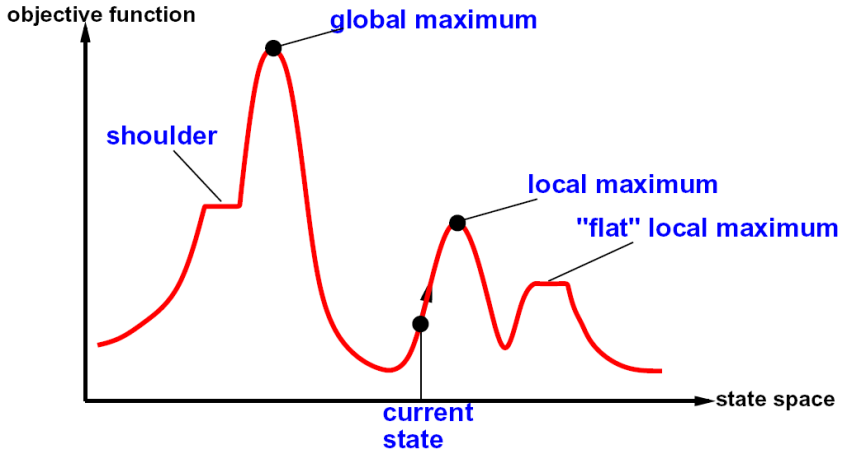
Hill Climbing

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
- What's good about it?

24

[Demo]

Hill Climbing Diagram



- Random restarts?
- Random sideways steps?

25

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```

26

Simulated Annealing

- Theoretical guarantee:
  - If T decreased slowly enough, simulated annealing will converge to the highest value (lowest cost) state!
- How useful is this guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape, the less likely you are to every make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

27