CS 188: Artificial Intelligence
Spring 2011

Lecture 4: A* + (beginnings of)
Constraint Satisfaction

1/31/2011

Pieter Abbeel – UC Berkeley

Many slides from Dan Klein and Max Likhachev

Announcements

- Project 1 (Search)
  - If you don't have a class account yet, pick one up after lecture
  - Still looking for project partners? --- Come to front after lecture
- Lecture videos
  - In the works

Today

- A* (tree) search
  - Admissible heuristics
- Graph search
  - Consistent heuristics
- Extensions
  - Weighted A*: f = g + eps h
  - Anytime A*
  - Memory issue (O(n)) → IDA*
  - Bi-directional
- Example Applications
- (Beginnings of CSPs)

Recap: Search

- Search problem:
  - States (configurations of the world)
  - Successor function: a function from states to lists of (state, action, cost) triples; drawn as a graph
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search Algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)

General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

Detailed pseudocode is in the book!

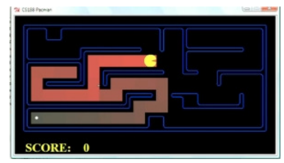
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

A* Review

- A* uses both backward costs g and forward estimate h: f(n) = g(n) + h(n)
- A* tree search is optimal with admissible heuristics (optimistic future cost estimates)
  - Proof forthcoming
- Heuristic design is key: relaxed problems can help
- Special cases:
  - Greedy: g = 0 [non-optimal!]
  - Uniform cost: h = 0 [optimal]

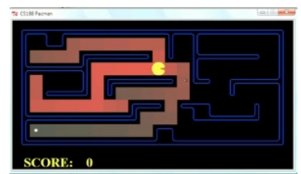
Comparison

Greedy



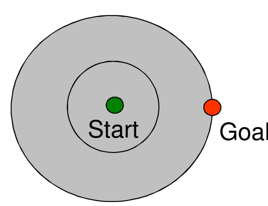
Uniform Cost

A star

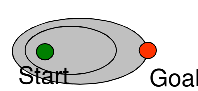


UCS vs A* Contours

- Uniform-cost expanded in all directions



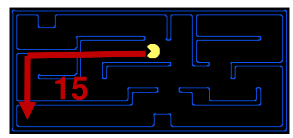
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

- Often, admissible heuristics are solutions to *relaxed problems,* with new actions (“some cheating”) available



- Inadmissible heuristics are often useful too (why?)

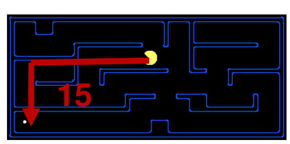
Admissible Heuristics

- A heuristic h is *admissible* (optimistic) if:

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

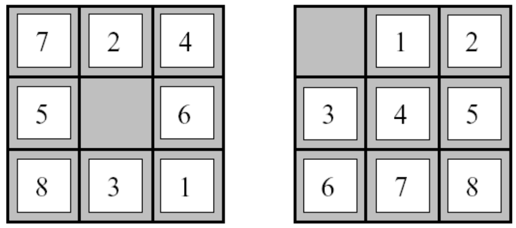
where $h^*(n)$ is the true cost to a nearest goal

- Example:



- Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Example: 8 Puzzle

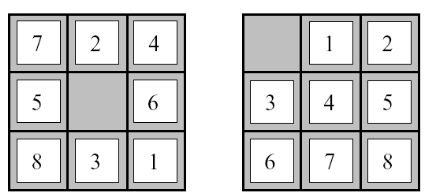


Start State Goal State

- What are the states?
- How many states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a relaxed-problem heuristic



Start State Goal State

	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why admissible?

- h(start) =

  3 + 1 + 2 + …

  = 18

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?

- With A*: a trade-off between quality of estimate and work per node!

Trivial Heuristics, Dominance

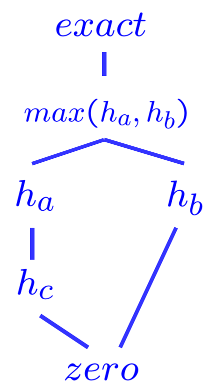
- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

$$h(n) = max(h_a(n), h_b(n))$$

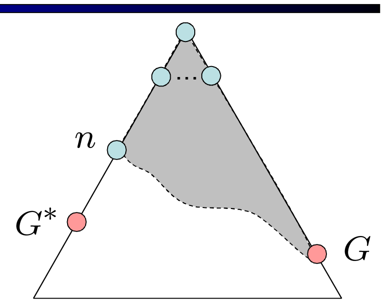
- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic



Optimality of A*: Blocking

Proof:

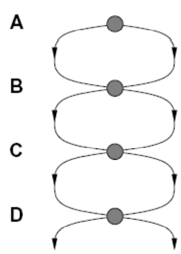
- What could go wrong?
- We'd have to have to pop a suboptimal goal G off the fringe before G*
- This can't happen:
  - Imagine a suboptimal goal G is on the queue
  - Some node *n* which is a subpath of G* must also be on the fringe (why?)
  - *n* will be popped before G

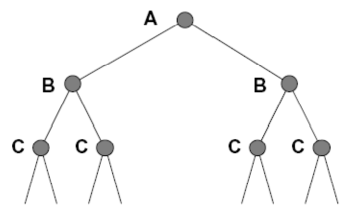


$$f(n) = g(n) + h(n)$$
$$g(n) + h(n) \leq g(G^*)$$
$$g(G^*) < g(G)$$
$$g(G) = f(G)$$
$$f(n) < f(G)$$

Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work. Why?





Graph Search

- Very simple fix: never expand a state twice

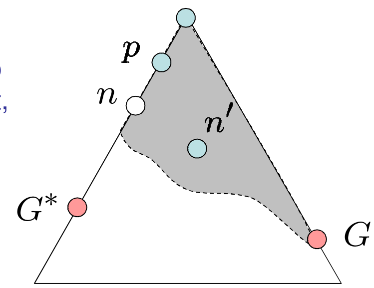
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
   closed ← an empty set
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
       if fringe is empty then return failure
       node ← REMOVE-FRONT(fringe)
       if GOAL-TEST(problem, STATE[node]) then return node
       if STATE[node] is not in closed then
           add STATE[node] to closed
           fringe ← INSERTALL(EXPAND(node, problem), fringe)
   end
```

- Can this wreck completeness? Optimality?

Optimality of A* Graph Search

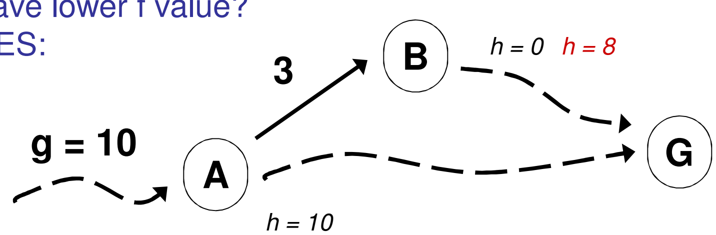
Proof:

- New possible problem: nodes on path to G* that would have been in queue aren't, because some worse *n'* for the same state as some *n* was dequeued and expanded first (disaster!)
- Take the highest such *n* in tree
- Let *p* be the ancestor which was on the queue when *n*' was expanded
- Assume *f(p) < f(n)*
- *f(n) < f(n')* because *n'* is suboptimal
- *p* would have been expanded before *n*'
- So *n* would have been expanded before *n*', too
- Contradiction!



Consistency

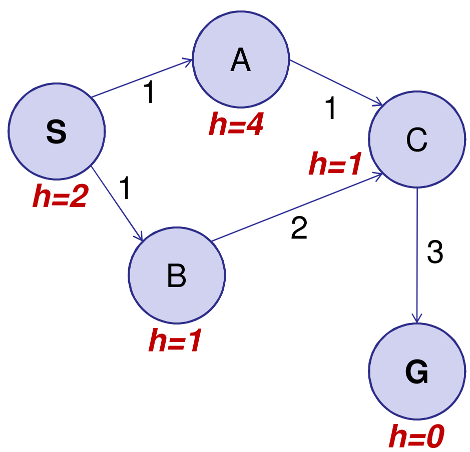
- Wait, how do we know parents have better f-vales than their successors?
- Couldn't we pop some node *n*, and find its child *n'* to have lower f value?
- YES:



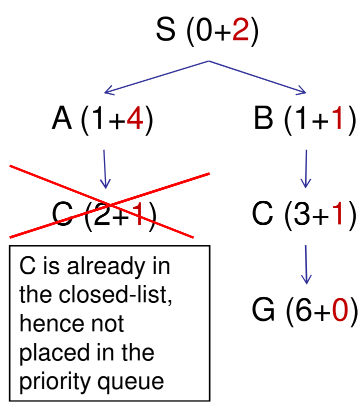
- What can we require to prevent these inversions?
- Consistency: $c(n, a, n') \geq h(n) - h(n')$
- Real cost must always exceed reduction in heuristic

A* Graph Search Gone Wrong

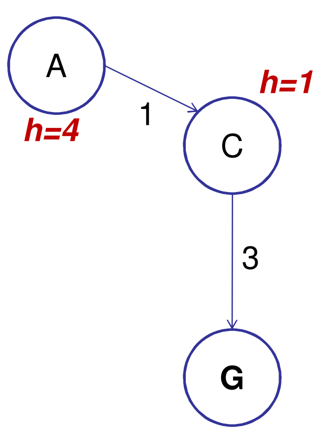
State space graph



Search tree



Consistency



The story on Consistency:

- Definition:
  cost(A to C) + h(C) ≥ h(A)
- Consequence in search tree:
  Two nodes along a path: $N_{A,}$ N_C
  $g(N_C) = g(N_A) + \text{cost}(A \text{ to } C)$
  $g(N_C) + h(C) \geq g(N_A) + h(A)$
- The f value along a path never decreases
- Non-decreasing f means you're optimal to every state (not just goals)

Optimality Summary

- Tree search:
  - A* optimal if heuristic is admissible (and non-negative)
  - Uniform Cost Search is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
  - Challenge:Try to prove this.
  - Hint: try to prove the equivalent statement *not admissible implies not consistent*
- In general, natural admissible heuristics tend to be consistent
- Remember, costs are always positive in search!

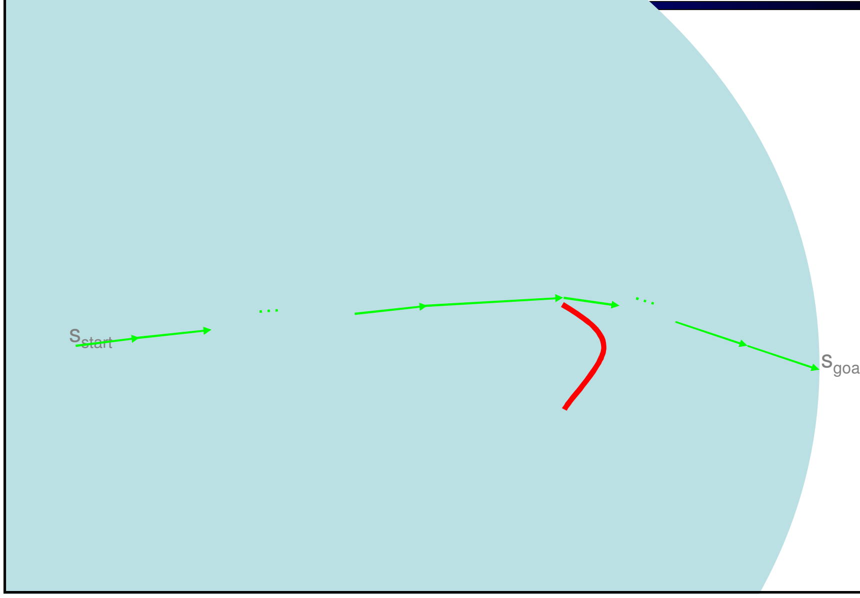
Today

- A* (tree) search
  - Admissible heuristics
- Graph search
  - Consistent heuristics
- Extensions
  - Weighted A*: f = g + eps h
  - Anytime A*
  - Memory issue (O(n)) → IDA*
  - Bi-directional
- Example Applications
- (Beginnings of CSPs)

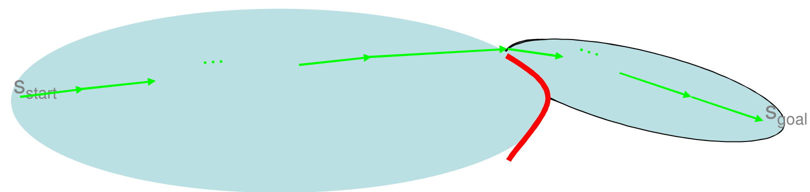
Weighted A* $f = g + \varepsilon h$

- **Weighted A*:** expands states in the order of
  $f = g + \varepsilon h$ values,
  $\varepsilon > 1$ = bias towards states that are closer to goal

Weighted A* $f = g + \varepsilon h$: $\varepsilon = 0$ --- Uniform Cost Search

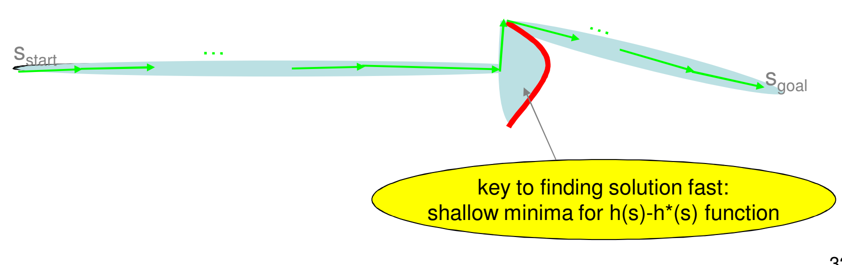


Weighted A* $f = g+\varepsilon h$: $\varepsilon = 1$ --- A*



31

Weighted A* $f = g+\varepsilon h$: $\varepsilon > 1$



32

Weighted A* $f = g + \varepsilon h : \varepsilon > 1$

- Trades off optimality for speed
- ε-suboptimal:
  - cost(solution) ≤ ε·cost(optimal solution)
  - Test your understanding by trying to prove this!
- In many domains, it has been shown to be orders of magnitude faster than A*
- Research becomes to develop a heuristic function that has shallow local minima

33

Anytime A*

- Weighted A*
  - Trades off optimality for speed
  - ε-suboptimal
- Anytime A*
  - For $\epsilon \in \{ \epsilon_1, \epsilon_2, \ldots, 1\}$
    - Run weighted A* with current ϵ
- [[ARA* and D*
  - efficient version of above that reuses state values within each iteration]]**

A* Memory Issues

- A* does provably minimum number of expansions (O(n)) for finding a provably optimal solution
- Memory requirements of A* (O(n)) can be improved though
- Memory requirements of weighted A* are often but not always better

35

A* Memory Issues → IDA*

- IDA* (Iterative Deepening A*)

  1. set $f_{max} = 1$ (or some other small value)
  2. execute (previously explained) DFS that does not expand states with $f>f_{max}$
  3. If DFS returns a path to the goal, return it
  4. Otherwise $f_{max}= f_{max}+1$ (or larger increment) and go to step 2

- Complete and optimal
- Memory: $O(bs)$, where b – max. branching factor, s – search depth of optimal path
- Complexity: $O(kb^s)$, where k is the number of times DFS is called

36

Bi-directional search

- If only 1 goal state:
  - Can simultaneously run two searches:
    - Search 1 starts at the START state
    - Search 2 starts at the GOAL state
  - → to find path from START to GOAL only requires two searches of depth s/2 rather than one of depth s
  - → $O(b^{(s/2)})$ vs. $O(b^s)$
- Challenge: think about how to run bidirectional A*

Robotics Examples

- Urban Challenge
  - Successor function?
  - Heuristic?
- Door Opening
  - Successor function?
  - Heuristic?

Other A* Applications

- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

Today

- A* (tree) search
  - Admissible heuristics
- Graph search
  - Consistent heuristics
- Extensions
  - Weighted A*: f = g + eps h
  - Anytime A*
  - Memory issue (O(n)) → IDA*
  - Bi-directional
- Example Applications
- (Beginnings of CSPs)

What is Search For?

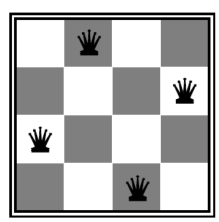
- Models of the world: single agents, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics to guide, fringe to keep backups

- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

41

Constraint Satisfaction Problems



- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test: any function over states
  - Successor function can be anything

- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

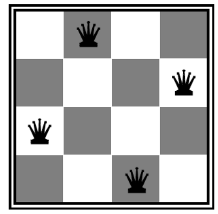
- Simple example of a *formal representation language*

- Allows useful general-purpose algorithms with more power than standard search algorithms

42

Example: N-Queens

- Formulation 1:
  - Variables: X_{ij}
  - Domains: $\{0, 1\}$
  - Constraints



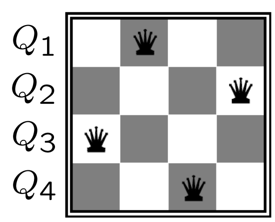
$$\forall i, j, k \ \ (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

43

Example: N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

-or-

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

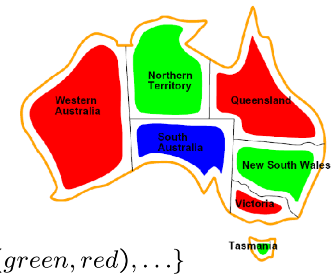
$\bullet \bullet \bullet$

Example: Map-Coloring

- Variables: $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$
- Domain: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors

$$WA \neq NT$$

$$(WA, NT) \in \{(red, green), (red, blue), (green, red), \ldots\}$$



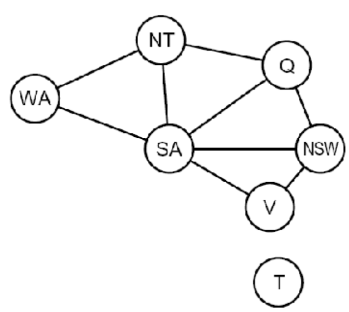
- Solutions are assignments satisfying all constraints, e.g.:

$$\{WA = red, NT = green, Q = red,$$
$$NSW = green, V = red, SA = blue, T = green\}$$

46

Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!



47

Example: Cryptarithmetic

- Variables (circles):

  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains:

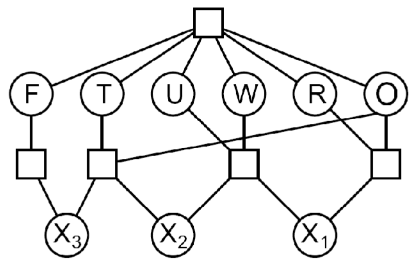
  $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints (boxes):

  $\text{alldiff}(F,T,U,W,R,O)$

  $O + O = R + 10 \cdot X_1$

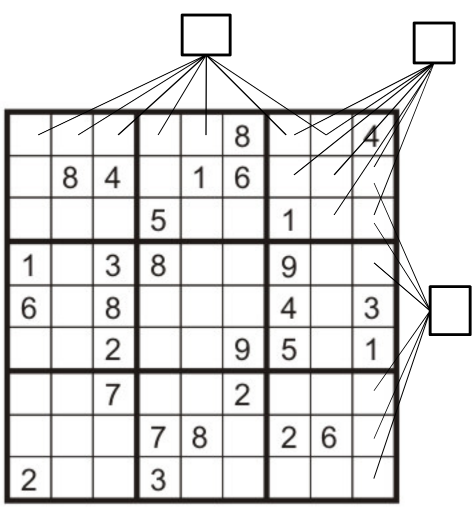
  $\cdots$

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



48

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

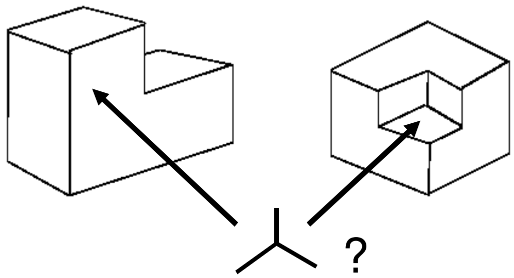
  9-way alldiff for each column

  9-way alldiff for each row

  9-way alldiff for each region

Example: The Waltz Algorithm

- The Waltz algorithm is for interpreting line drawings of solid polyhedra
- An early example of a computation posed as a CSP



- Look at all intersections
- Adjacent intersections impose constraints on each other

50

Varieties of CSPs

- Discrete Variables
  - Finite domains
    - Size d means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start-end state of a robot
  - Linear constraints solvable in polynomial time by LP methods (see cs170 for a bit of this theory)

54

Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equiv. to shrinking domains):

    $$SA \neq green$$

  - Binary constraints involve pairs of variables:

    $$SA \neq WA$$

  - Higher-order constraints involve 3 or more variables:
    e.g., cryptarithmetic column constraints

- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We'll ignore these until we get to Bayes' nets)

55

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floorplanning
- Fault diagnosis
- … lots more!

- Many real-world problems involve real-valued variables…

56