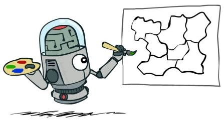
Artificial Intelligence

Module 3: Search Strategies

PART 3.6: Constraint Satisfaction Problems



Dr. Chandra Prakash

Assistant Professor

Department of Computer Science and Engineering

(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Dan Klein and Pieter Abbeel, PP Chakrabarti

Module 3: Search Strategies

- PART 3.1: Search
- PART 3.2: Uninformed Search
- PART 3.3: Informed/Heuristic Search
- PART 3.4: Beyond Classical Search
  - Local Search
    - Generate-and-test
    - Hill climbing
    - Simulated Analing
  - Problem reduction
- PART 3.5: Adversarial Search
- PART 3.6: Constraint Satisfaction Problems

Exercise : Find the Similarity

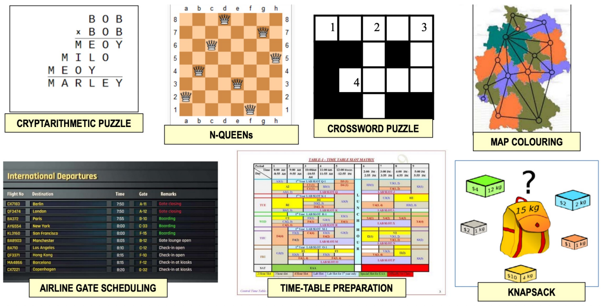
Plagiarism or not ??

- M1 Respected Sir, I'm extremely sorry as I couldn't submit my 4_5 Assignment in time. My assignment was completed just in time. But I couldn't submit as it stopped taking responses immediately after the deadline. I'd be really grateful if you could please allow me to submit the assignment. I won't repeat this late-submission in future.
- M2 Respected Sir, I am extremely sorry that I couldn't submit the 4&5 Assignment before the deadline. Initially the deadline was of 22 ' September (I don't have Team App in laptop due to some issue. I am using teams in browser. So i did get any notification and I was not aware about the prepone of the deadline. Sir I have already shown by code for question 1 & questions 2 in class. I have completed my assignment. Sir, could you please allow me to submit the TUT ? I will be very thankful to you. And I assure you that I won't repeat this mistake.
- M3 Respected Sir, I am extremely sorry that I could not submit the Assignment 4&5 before the deadline. I completed my work around 1 AM at night. I was busy giving my internship exam . Sir, could you please allow me to submit the TUT ? I will be very thankful to you. And I assure you that I won't repeat this mistake. Thanks & Regards.
- M4 Respected Sir, I am extremely sorry that I could not submit the 4&5 before the deadline. I completed my work around 2 AM at night. Because I was busy in Aptitude , Verbal , Computer Knowledge & Coding rounds of several internships at D2C platform for two days (Saturday and Sunday). Sir, could you please allow me to submit the TUT in the late submission? I will be very thankful to you sir. And I assure you I won't repeat this mistake ever again. Thanks & Regards.

Problem formulation for Similarity /Plagiarism

Problem formulation for Deadlines

Consistent Labelling by Constraint Satisfaction



Constraint Satisfaction Problems (CSPs)

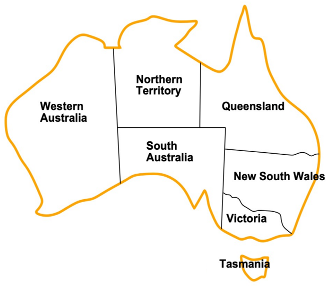
- **Standard search problem:**
  - state is a "black box"—any old data structure that supports goal test, eval, successor
  - Goal test and Successor can be any function over states
  - Part of representation of atomic state
- **Constraint satisfaction problems (CSPs):**
  - A special subset of search problems
  - state is defined by variables X_i with values from domain D_i ((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
- **AI as search → AI as Represenation** → AI as ML Mindset
  - Constraint Satisfaction Problem
  - Preposition Logic
  - Bayesian network

Example: Class

- Representation of atomic state

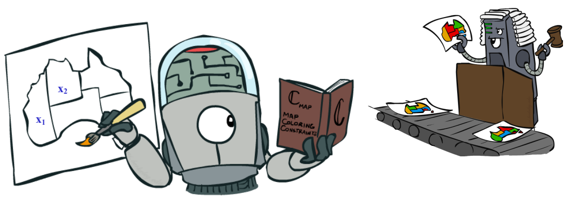
- CSP
  - Understand the problem
  - Now we need to understand the structure of the state i.e class
  - factored representation of each state
    - set of variable and each variable has a value

CSP Examples: Map-Coloring



Constraint Satisfaction Problems

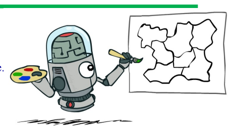
N variables
domain D
constraints

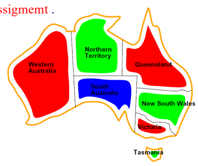


states	goal test	successor function
partial assignment	complete; satisfies constraints	assign an unassigned variable

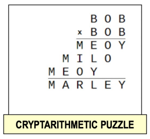
Example: Map Coloring

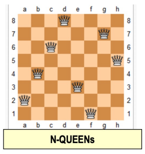
- **Variables**: A Finite Set of Variables (V_1, V_2, …., V_n)
  - `WA, NT, Q, NSW, V , SA, T`
- **Domains**:
  - Each Variable has a Domain D_1, D_2, …., D_n from which it can take a value.
  - The Domains may be discrete or continuous domains
  - D_i = {red, green, blue}
- **Constraints**:
  - A Finite Set of Satisfaction Constraints, C_1, C_2, …C_m
  - An assignment that does not violate any constraints is called a consistent or legal assigmemt .
  - adjacent regions must have different colors
  - Implicit:
    - WA ≠NT (if the language allows this)
  - Explicit:
    - *(WA,NT)∈{(red,green),(red,blue),(green,red),. . .}*
- **Optimization Criteria (Optional)**
  - A Set of Optimization Functions O_1, O_2, ….O_p
- **Solutions :**
  - a consistent and complete assignment.
  - are assignments satisfying all constraints, and the optimization criteria if any e.g.:
  - *{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green}*

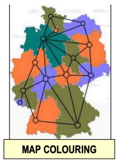




Formulating CSPs







1. VARIABLES
2. DOMAINS
3. SATISFACTION CONSTRAINTS
4. OPTIMIZATION CRITERIA
5. SOLUTION

Why CSP ??

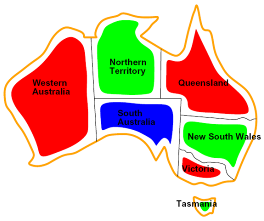
- CSPs yield a natural representation for a wide variety of problems
  - if you already have a CSP-solving system, it is often easier to solve a problem using it than to design a custom solution using another search technique.
- CSP solvers can be faster than state-space searchers because the CSP solver can quickly eliminate large swatches of the search space.
  - For example, once we have chosen {SA = blue} in the Australia problem, we can conclude that none of the five neighboring variables can take on the value blue.
  - Without taking advantage of constraint propagation, a search procedure would have to consider $3^5 = 243$ assignments for the five neighboring variables; with constraint propagation we never have to consider blue as a value, so we have only $2^5 = 32$ assignments to look at, a reduction of 87%.

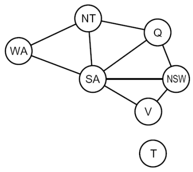
CSP Solution Overview

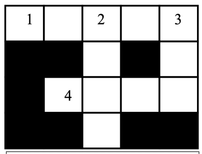
- **CSP Graph Creation:**
  - Create a Node for Every Variable. All possible Domain Values are initially Assigned to the Variable
  - Draw edges between Nodes if there is a Binary Constraint. Otherwise Draw a hyper- edge betwee nodes with constraints involving more than two variables
- **Constraint Propagation:**
  - Reduce the Valid Domains of Each Variable by Applying Node Consistency, Arc / Edge Consistency, K-Consistency, till no further reduction is possible.
  - If a solution is found or the problem found to have no consistent solution, then terminate
- **Search for Solution:**
  - Apply Search Algorithms to Find Solutions
  - There are interesting properties of CSP graphs which lead of efficient algorithms in some cases: Trees, Perfect Graphs, Interval Graphs, etc
  - Issues for Search: Backtracking Scheme, Ordering of Children, Forward Checking (Look-Ahead) using Dynamic Constraint Propagation
  - Solving by Converting to Satisfiability (SAT) problems

Exercise : Draw CSP Graph

*It can be helpful to visualize a CSP as a **constraint graph***



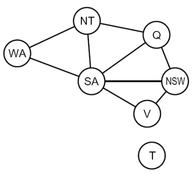




Word List:
astar, happy, hello, hoses, live, load, loom, peal, peel, save, talk, ant, oak, old

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!
- Possible only when we understand the structure of the problem
  - each state variable is a node not a state, earlier each state was a node.



Varieties of CSPs

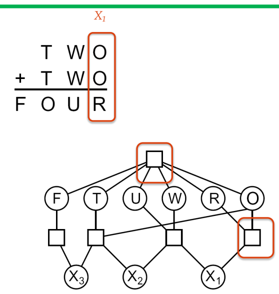
- Discrete variables
  - finite domains; size d ⇒ $O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
  - infinite domains (integers, strings, etc.)
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
    - 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

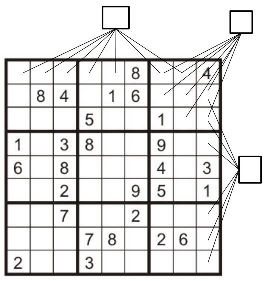
- Unary or Node constraints
  - involve a single variable,
  - e.g., SA ≠ green
- Binary or Edges constraints
  - involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order or Hyper-Edge constraints
  - involve 3 or more variables,
  - e.g., cryptarithmetic column constraints
- Preferences (soft constraints),
  - e.g., red is better than green
  - often representable by a cost for each variable assignment → constrained optimization problems (We'll ignore these until we get to Bayes' nets)

Example: Cryptarithmetic

- Variables:
  - F T U W R O X1 X2 X3
- Domains:
  - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints:
  - alldiff(F, T,U, W, R, O)
  - O + O = R + 10 · X1, etc.
  - . . .



Example: Sudoku



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

  9-way alldiff for each column

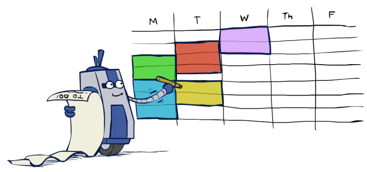
  9-way alldiff for each row

  9-way alldiff for each region

  (or can have a bunch of pairwise inequality constraints)

Real-World CSPs

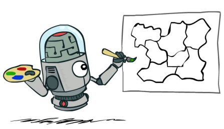
- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!



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

Solving CSPs: Mapping problem

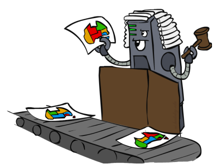
- Initial State :
  - not assigned any variable whatsoever
- Sucessor variable:
  - assigned *l* variables
  - *n-l* options for assignment
  - each variable possible d values
  - children :
    - *n-l* d*
- Solution is at :
  - *n* depth
- How many leaves do we have:
  - $n*d + (n-1)*d+(n-2)*d + \ldots\ldots = n!d^n$



Standard Search Formulation (incremental)

- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment.
    - ⇒ fail if no legal assignments (not fixable!)
  - Goal test: the current assignment is complete and satisfies all constraints

  1) This is the same for all CSPs!
  2) Every solution appears at depth n with n variables
     ⇒ use depth-first search
  3) Path is irrelevant, so can also use complete-state formulation
  4) $b = (n - l)d$ at depth l, hence $n!d^n$ leaves!!!!



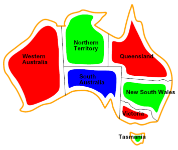
- We'll start with the straightforward, naïve approach, then improve it

Search Methods

- What would BFS do?

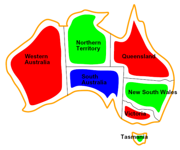
{}

{WA=g} {WA=r} {WA=b} {NT=g} …



Search Methods

- What would BFS do?
- What would DFS do?
  - let's see!
- What problems does naïve search have?



Video of Demo Coloring -- DFS

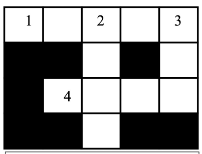
CSP

- In regular state-space search, an algorithm can do only one thing: *search*.
- In CSPs there is a choice:
  - an algorithm can search (choose a new variable assignment from several possibilities) or
  - do a specific type of **inference** called **constraint propagation**:
    - using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on
- Constraint propagation may be intertwined with search, or it may be done as a preprocessing step, before search starts.

2. Constraint Propagation Steps

- **Constraints**
  - Unary Constraints or Node Constraints
  - Binary Constraints or Edges between CSP Nodes
  - Higher order or Hyper-Edges between CSP Nodes
- **Node Consistency**
  - For every Variable V_i, remove all elements of D_i that do not satisfy the Unary Constraints for the Variable
  - First Step is to reduce the domains using Node Consistency
- **Arc Consistency**
  - For every element x_{ij} of D_i, for every edge from V_i to V_j, remove x_{ij} if it has no consistent value(s) in other domains satisfying the Constraints
  - Continue to iterate using Arc Consistency till no further reduction happens.
- **K-Consistency or Path Consistency**
  - For every element y_{ij} of D_i, choose a Path of length L with L variables, use a consistency checking method similar to above to reduce domains if possible
- **Global constraints**

CSP Graph for Crossword



Word List:
astar, happy, hello, hoses, live, load, loom, peal, peel, save, talk, ant, oak, old

Applying Node Consistency:
D1 = {astar, happy, hello, hoses}
D2 = {live, load, loom, peal, peel, save, talk}
D3 = {ant, oak, old}
D4 = {live, load, loom, peal, peel, save, talk}

NOW APPLY ARC CONSISTENCY

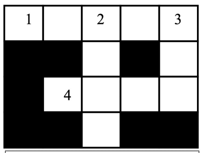
Applying Arc Consistency:
D1 = {astar, happy, hello, hoses}

D2 = {live, load, loom, peal, peel, save, talk}

D3 = {ant, oak, old}

D4 = {live, load, loom, peal, peel, save, talk}

CSP Graph for Crossword



Word List:
astar, happy, hello, hoses, live, load, loom, peal, peel, save, talk, ant, oak, old

Applying Node Consistency:
D1 = {astar, happy, hello, hoses}
D2 = {live, load, loom, peal, peel, save, talk}
D3 = {ant, oak, old}
D4 = {live, load, loom, peal, peel, save, talk}

NOW APPLY ARC CONSISTENCY
D1 - D2 , D1 – D3 , D2-D4, D3-D4

Applying Arc Consistency:
D1 = {~~astar~~, ~~happy~~, hello, ~~hoses~~}

D2 = {~~live~~, load, loom, peal, ~~peel~~, ~~save~~, ~~talk~~}

D3 = {~~ant~~, oak, old}

D4 = {~~live~~, load, ~~loom~~, ~~peal~~, ~~peel~~, ~~save~~, talk}

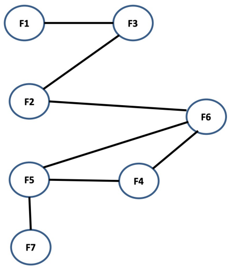
Arc Consistency Algorithm AC-3

AC-3(*csp*) // inputs - CSP with variables, domains, constraints
queue ← local variable initialized to all arcs in csp
while *queue* is not empty **do**
  (X_i, X_j) ← pop(queue)
  if Revise(*csp*, X_i, X_j) **then**
    if size of D_i = 0 **then return** *false*
    for each X_k **in** X_i.neighbors-{X_j} **do**
      add (X_k, X_i) to *queue*
return *true*

Revise(*csp*, X_i, X_j)
revised ← *false*
for each x **in** D_i **do**
  if no value y in D_j allows (x, y) to satisfy constraint between X_i and X_j **then**
    delete x from D_i
    revised ← *true*
return *revised*

Solve: Airline Gate Scheduling

Flight No	Dep Time	G Start	G End
F1	7:00	6:15	7:15
F2	8:30	7:45	8:45
F3	7:45	7:00	8:00
F4	9:45	9:00	10:00
F5	10:00	9:15	10:15
F6	9:00	8:15	9:15
F7	11:00	10:15	11:15



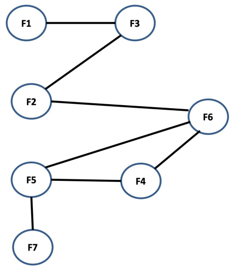
- Three gates { 1,2,3}
- Minimize gate Dom : 7 Gates

Consistency for Airline Gate Scheduling

- Three gates { 1,2,3}

Flight No	Dep Time	G Start	G End
F1	7:00	6:15	7:15
F2	8:30	7:45	8:45
F3	7:45	7:00	8:00
F4	9:45	9:00	10:00
F5	10:00	9:15	10:15
F6	9:00	8:15	9:15
F7	11:00	10:15	11:15

- {1,2}



Backtracking Algorithm for CSP

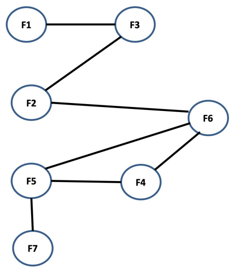
CSP-BACKTRACKING({})

CSP-BACKTRACKING(a)
- If a is complete then return a
- X ← select unassigned variable
- D ←select an ordering for the domain of X
- For each value v in D do
  - If v is consistent with a then
    - Add (X= v) to a
    - result ← CSP-BACKTRACKING(a)
    - If result ≠ failure then return result
- Return failure

Backtracking for Airline Gate Scheduling

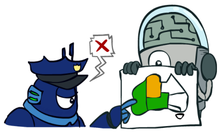
- Two gates { 1,2}

Flight No	Dep Time	G Start	G End
F1	7:00	6:15	7:15
F2	8:30	7:45	8:45
F3	7:45	7:00	8:00
F4	9:45	9:00	10:00
F5	10:00	9:15	10:15
F6	9:00	8:15	9:15
F7	11:00	10:15	11:15



Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- **Idea 1:** One variable at a time
  - Variable assignments are commutative, so fix ordering -> better branching factor!
  - I.e., *[WA = red then NT = green]* same as *[NT = green then WA = red]*
  - Only need to consider assignments to a single variable at each step
  - ⇒ b = d and there are d^n leaves
- **Idea 2:** Check constraints as you go
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"



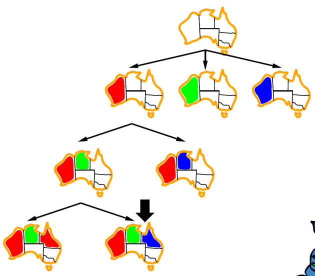
- Backtracking search
  - Depth-first search with these two improvements is called *backtracking search*
  - *Depth-first search for CSPs with single-variable assignments*
- Can solve n-queens for n ≈ 25

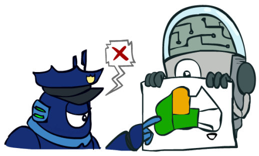
Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given CONSTRAINTS[csp] then
         add {var = value} to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failure then return result
         remove {var = value} from assignment
   return failure
```

- Backtracking = DFS + variable-ordering + fail-on-violation
- What are the choice points?

Backtracking : Map Coloring Example

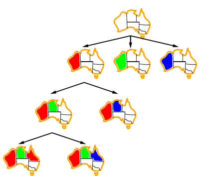




Video of Demo Coloring – Backtracking

Improving Backtracking efficiency

- Initial Constraint Propagation
- General-purpose ideas give huge gains in speed
- **Ordering:**
  - Which variable should be assigned next?
    - Most Constrained Variable / Minimum Remaining Values
    - Most Constraining Variable
  - In what value / order should its values be tried?
    - Least Constraining Value leaving maximum flexibility
- **Filtering:**
  - Can we detect inevitable failure early?
    - Preventing useless Search ahead
  - Can we take advantage of problem structure?
- **Dependency Directed Backtracking**
- **Other CSP Search Algorithms :**
  - SAT Formulations and Solvers
  - Optimization
    - Branch-and-Bound
    - SMT Solvers, Constraint Programming
  - Learning, Memoizing,CSP Problems are NP-Hard in Genera



Ordering: Minimum Remaining Values (MRV)

- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain

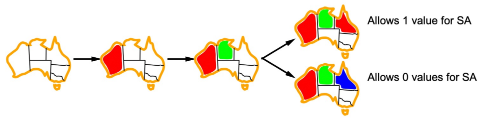
- Why min rather than max?
- Also called "most constrained variable"
- "Fail-fast" ordering

Degree heuristic

- Tie-breaker among MRV variables
- Degree heuristic:
  - choose the variable with the most constraints on remaining variables

Ordering: Least Constraining Value

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the *least constraining value*
    - the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)



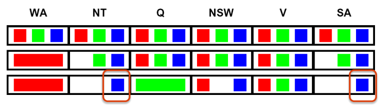
- Why least rather than most?
- Combining these ordering ideas makes 1000 queens feasible

Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking:
  - Forward checking propagates information from assigned to unassigned variables
  - Cross off values that violate a constraint when added to the existing assignment
  - Terminate search when any variable has no legal values

Filtering: Constraint Propagation

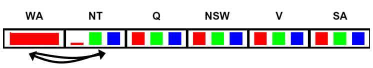
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

Consistency of A Single Arc

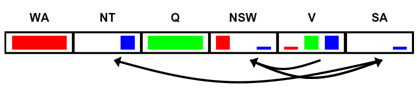
- Simplest form of propagation makes each arc consistent
- An arc X → Y is consistent iff
  - for *every* x in the (X) tail there is *some* y in the head
  - which could be assigned without violating a constraint



Forward checking?
Enforcing consistency of arcs pointing to each new assignment

Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:



- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
- What's the downside of enforcing arc consistency?

Remember: Delete from the tail!

Enforcing Arc Consistency in a CSP

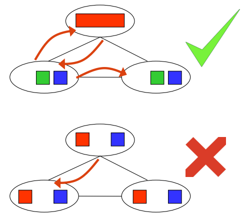
```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) <- REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed <- false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi <-> Xj
         then delete x from DOMAIN[Xi]; removed <- true
   return removed
```

- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

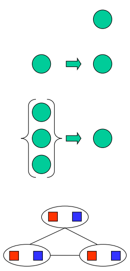
Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)
- Arc consistency still runs inside a backtracking search!



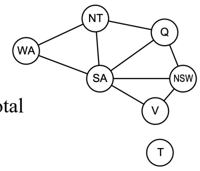
K-Consistency

- 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 case: arc consistency)



Problem Structure

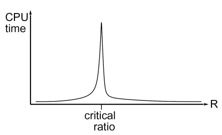
- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph
- Suppose each subproblem has c variables out of n total
- Worst-case solution cost is $n/c \cdot d^c$, linear in n
- E.g.,
  - n = 80, d = 2, c = 20
  - 2^{80} = 4 billion years at 10 million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

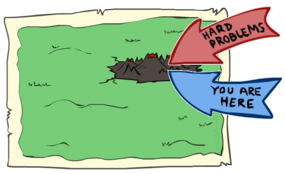


Performance of Min-Conflicts

- 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

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



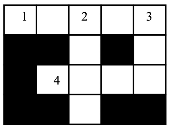


Summary

- 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 guarantee later failure
- Constraint propagation (e.g., arc consistency) 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

Home Work : Formulate CSP

- Crossword
- Flight Gate Scheduling
- Mention
  - 1. VARIABLES
  - 2. DOMAINS
  - 3. SATISFACTION CONSTRAINTS
  - 4. OPTIMIZATION CRITERIA
  - 5. SOLUTION
  - CSP Graph
    - Apply Node Consistency
    - Apply Arc Consistency



Word List:
astar, happy, hello, hoses, live, load, loom, peal, peel, save, talk, ant, oak, old

1. VARIABLES
2. DOMAINS
3. SATISFACTION CONSTRAINTS
4. OPTIMIZATION CRITERIA
5. SOLUTION

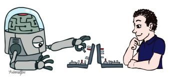
Flight No	Dep Time	G Start	G End
F1	7:00	6:15	7:15
F2	8:30	7:45	8:45
F3	7:45	7:00	8:00
F4	9:45	9:00	10:00
F5	10:00	9:15	10:15
F6	9:00	8:15	9:15
F7	11:00	10:15	11:15

Next :

- Module 4: Logic and Deduction

References

- ***Artificial Intelligence*** *by Elaine Rich & Kevin Knight*, Third Ed, Tata McGraw Hill
- ***Artificial Intelligence and Expert System*** *by Patterson*
- http://www.cs.rmit.edu.au/AI-Search/Product/
- http://aima.cs.berkeley.edu/demos.html (for more demos)
- ***Artificial Intelligence and Expert System*** *by Patterson*
- Slides adapted from CS188 Instructor: Anca Dragan, University of California, Berkeley
- Slides adapted from CS60045 ARTIFICIAL INTELLIGENCE



(some slides adapted from http://aima.cs.berkeley.edu/)