CS5100: Foundations of Artificial Intelligence

Constraint Satisfaction Problems (CSP)

Dr. Rutu Mulkar-Mehta
Lecture 2

Some slides and images used from Berkeley CS188 course notes, with permission

Administrative

- Assignment P0 – How many did it?
- Project 1 - How many started it?
  - If you have not started it, please start now
  - Submission details to follow next week
- Last week's in-class assignment – Will be graded and handed back to you next week
  - If you are not happy with your performance, don't worry, I will drop your lowest scoring assignment
  - If you have questions about how your assignment was graded, contact me
- Today – We have an optional in class assignment for "Extra credit". If you do not plan to work on it, you may leave early

Search: Review

- Informed Search
  - DFS, BFS, UCS, Iterative Deepening etc.
- Uninformed Search
  - Best First Search
  - Greedy Best First Search
  - A* Search

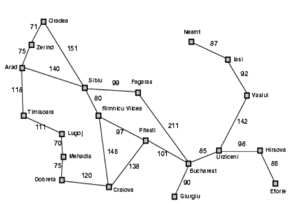
3

Limitations of Simple Search

- State is considered a "black box"
  - a data structure that supports
    - successor function – f(n)
    - heuristic function – h(n)
    - goal test
- What if we had some constraints on states of our problem?

4

Constraint Examples



- Eg. constraints:
  - Sibiu only allows traffic from Oradea
  - Pitesti has a lot of construction going on, avoid it
  - Fagaras charges tolls, avoid it

5

Outline for Today

- Constraint Satisfaction Problems (CSP)
- Solving CSP's
  - Backtracking search

6

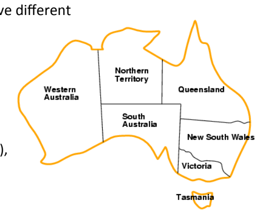
Constraint satisfaction problems (CSPs)

- Factored Representation of each state.
  - Each state has variables that can take a value
- Use general purpose rather than problem specific heuristics
- Components
  - $X_{1..n}$: A set of Variables
  - $D_{1..n}$: A set of Domains
  - C : A set of constraints that specify allowable combination of values
    - *<scope, rel>*

7

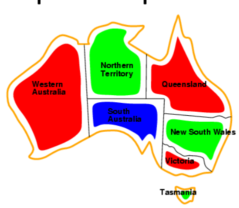
Example: Map-Coloring

- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red, green, blue}
- Constraints: adjacent regions must have different colors
  - How many constraints do we have?
- e.g.,
  - WA ≠ NT
  - (WA,NT) in {(red, green), (red, blue), (green, red), (green, blue), (blue, red),(blue, green)}



8

Example: Map-Coloring

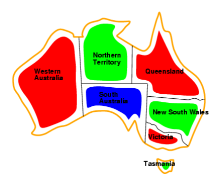


- e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

9

How are CSP's evaluated?

- Complete
  - One in which every variable is assigned a solution
- Consistent
  - One in which no constraints are violated



10

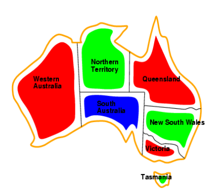
Why formulate a problem as a CSP?

- CSP's are a natural representation of a wide variety of problems
- CSP's are generic – if you have an implementation of one CSP, you can use it to solve different problems
- CSP's are faster problem solvers, as they quickly eliminate a large number of state spaces

11

CSP's are fast Problem Solvers

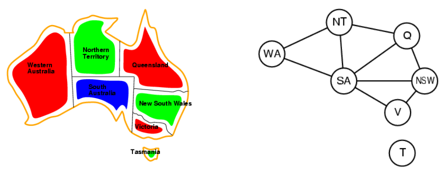
- If SA = *Blue*, none of the other neighbors can be blue
- Search Complexity?
  - 3^5 Next States = 243
- CSP Complexity?
  - 2^5 Next States = 32
- CSP has a reduction of 87%



12

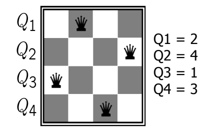
Constraint graph

- Constraint graph: nodes are variables, arcs are constraints
- Binary CSP: each constraint relates two variables



Example: N-Queens

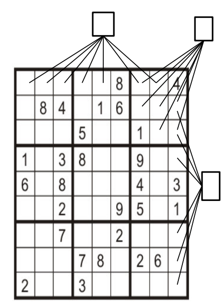
– Variables: Q_k
  (k is the row)
– Domains: $\{1, 2, 3, \ldots N\}$
  (These are the columns)
– Constraints:
  Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)
  – Q_i != Q_j (cannot be in the same row)
  – $|Q_i - Q_j|$!= $|i-j|$ (cannot be in the same diagonal)
  Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$
  …



Scheduling Constraints: Job-Shop

- Scheduling a car assembly: (15 Tasks)
  – Install Axles (front and back)
  – Install Wheels (4 wheels)
  – Tighten nuts for each wheel
  – Affix Hubcaps
  – Inspect Final assembly
- Constraints: Task Dependencies
  – e.g. Axels must be installed before wheel
  – e.g. All assemble must be done before final inspection
  – e.g. $Axle_F + 10 <= Wheel_{RF}$
  – (Front Axel + 10 minutes <= Rear Front wheel)

Example: Sudoku



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

How many Variables, Domains and 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)

Variations of CSP Formalisms

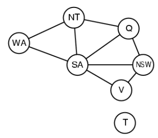
- Variables: Discrete vs Continuous
  – Map coloring?
  – N-Queens?
  – Scheduling?
- Domains: Finite vs Infinite
  - Map coloring?
  – N-Queens?
  – Scheduling?

Variations of CSP Formalisms

- Discrete variables and Finite Domains
  – *n:* variables, d: domain
  – *How many assignments?*
  – *$O(d^n)$* complete assignments
- Discrete Variables and Infinite domains:
  – integers, strings, etc.
  – e.g., job scheduling, variables are start/end days for each job
  – need a constraint language, e.g., *$StartJob_1 + 5 \leq StartJob_3$*
- Continuous variables
  – e.g., start/end times for Hubble Space Telescope observations
  – linear constraints solvable in polynomial time by linear programming

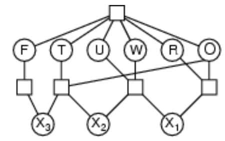
Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order (Global Constraint) constraints involve 3 or more variables,
  - e.g., *alldiff* in Sudoku,
  - e.g. cryptarithmetic column constraints



Example: Cryptarithmetic

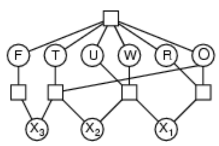
```
  T W O
+ T W O
F O U R
```



- Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains: {$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$}
- Constraints: *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F, T \neq 0, F \neq 0$

Constraint Hypergraph

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



- Nodes – Circles
- Hyper nodes – Square (represent *n*-ary constraints)

Dual Graph Transformation

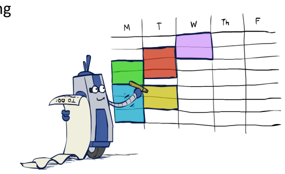
- The process of transforming an n-ary relation into a binary relation
- This is however not preferred – why?
  - Less error prone and easier to implement global constraints like *alldiff*
  - Special complex algorithms for higher order constraints that might not be available for lower order constraints

Preference Constraints

- They are not required constraints, but preferred constraints
  - e.g. avoiding toll roads is not required, but preferred
- Encoded as costs on individual variable assignments. Higher costs make a more expensive path and is not preferred
- We want our Objective Function to reduce the overall costs
- This is the Constraint Optimization Problem

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!

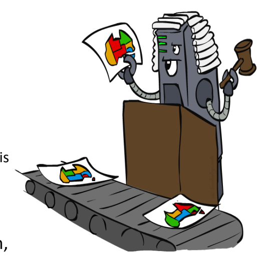


- Many real-world problems involve real-valued variables…

Solving CSPs

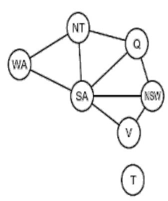
Standard Search Formulation

- States defined by the values assigned so far (partial assignments)
  - Initial state:
    - the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints
- We'll start with the straightforward, naïve approach, then improve it



Search Methods

- What would BFS do?
- What would DFS do?
- What problems does naïve search have?



Naïve Solution: Apply BFS, DFS, A*…

_ _ _ _ _ _ _

R _ _ _ _ _ _

R G _ _ _ _ _

R G R _ _ _ _

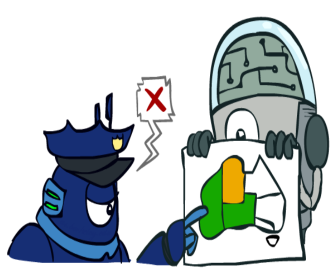
⋮

R G R R R R R

How many leaf nodes are expanded in the words case?
- How many Variables?
- How many Domains?

28

Backtracking Search



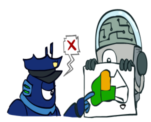
Backtracking Search

- The term Backtracking Search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign

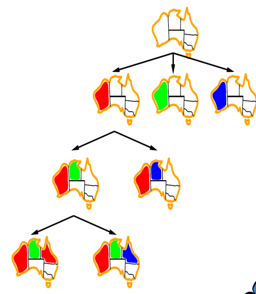
30

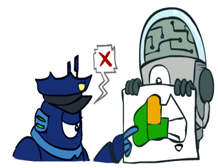
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - 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
- 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"
- Depth-first search with these two improvements is called *backtracking search*
- Can solve n-queens for n ≈ 25



Backtracking Example





Backtracking Search

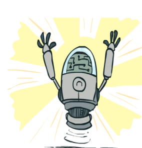
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for 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

Video of Demo Coloring – Backtracking

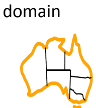
Improving Backtracking

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failure early and filter out bad states?



Ordering: Minimum Remaining Values

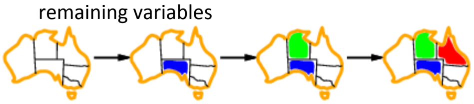
- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain



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

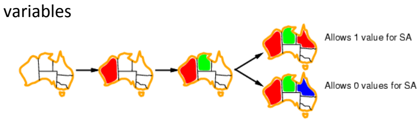
Ordering: Degree Heuristic

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



Ordering: Least Constraining Value

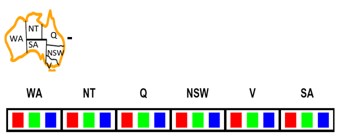
- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables



- Combining these heuristics makes 1000 queens feasible

Filtering: Forward Checking

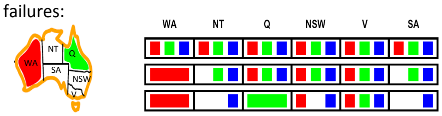
- Keep track of domains for unassigned variables and cross off ba options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment



Video of Demo Coloring – Backtracking with Forward Checking

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

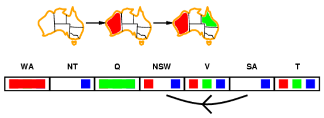
Constraint Propagation

- Node Consistency
  - A single variable corresponding to a node in the CSP network is node consistent, if all the values in the variable's domain satisfy the variables unary constraints
- Arc Consistency
  - A variable in CSP is arc consistent if every value in its domain satisfies the variables binary constraints
  - e.g. $Y = X^2$
  - e.g. Map Coloring Problem

Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

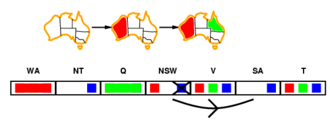
for every value *x* of *X* there is some allowed *y*

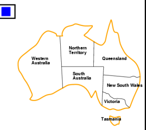


Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

for every value *x* of *X* there is some allowed *y*

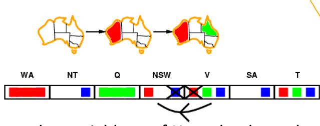




Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

for every value *x* of *X* there is some allowed *y*



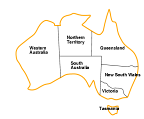
- If *X* loses a value, neighbors of *X* need to be rechecked

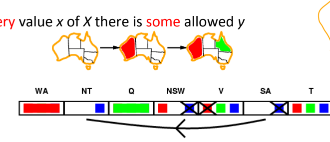
45

Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

for every value *x* of *X* there is some allowed *y*





- 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

46

Enforcing Arc Consistency in a CSP

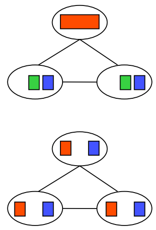
```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  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
```

Video of Demo Arc Consistency – CSP Applet – n Queens

Limitations of Arc Consistency

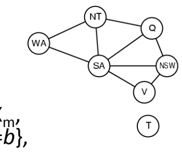
- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)
- Arc consistency still runs inside a backtracking search!



What went wrong here?

Constraint Propagation

- Path Consistency
  - What if we have 2 colors for Map coloring problem? *(blue, red)*
  - Is it arc consistent? – Yes
  - Does that help? – No
- $\{X_i, X_j\}$ are path consistent with X_m, if for every assignment $\{X_i = a, X_j = b\}$, there is an assignment that satisfied X_m
  - e.g. blue and red colors for Map problem



50

Video of Demo Coloring – Backtracking with Forward Checking – Complex Graph

Video of Demo Coloring – Backtracking with Arc Consistency – Complex Graph

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
- Iterative min-conflicts is usually effective in practice

53