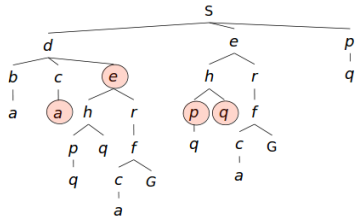
Today.

http://bit.ly/3GEMokW
Graph Search.
Consistent Heuristic.

Constraint Satisfaction Problems.

Graph Search

http://bit.ly/3GEMokW



In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

Graph Search

Idea: never expand a state twice

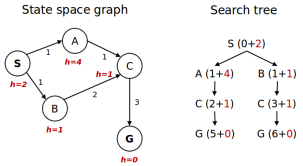
How to implement:
Tree search + set of expanded states ("closed set")
Expand the search tree node-by-node, but...
Before expanding a node,
check if state was never been expanded before
If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness? Why/why not?

How about optimality?

A* Graph Search Gone Wrong?



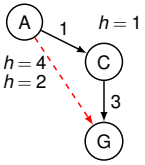
Is $h(\cdot)$ admissible? Yes.

Will exploring w.r..t $h(B)+g(n)$ be optimal?

Expand S.
A and B in fringe!
Expands B, since $h(B)+g(B)=2<5=h(A)+g(A)$.
C in fringe with key, $3+h(C)=4$.
G in fringe with key, 5.

Could have been there in 4.

Consistency of Heuristics



Main idea: est. heuristic costs $\leq$ actual costs
Admissibility: $h(x) \leq$ cost to goal.
Consistency: $h(x)-h(y) \leq cost(x,y)$.
heuristic "arc" cost $\leq$ actual arc cost

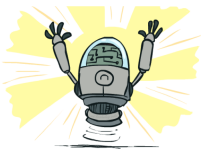
Consistent $\Longrightarrow$ admissible? Yes? No?

Consistent: f value along a path never decreases
Admissible:
$f(C)=h(C)+1=3$.
$f(A)=h(A)=4$.
Consistent: $f(A)=2<3=f(C)$.

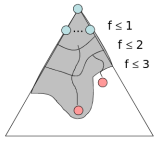
Claim: If y is expanded due to x, $f(y) \geq f(x)$.
Proof:
$f(y)=g(x)+cost(x,y)+h(y)$
$\geq g(x)+h(x)-h(y)+h(y)=g(x)+h(x)=f(x)$ □

The "estimate" of plan cost keeps rising as you progress.

Optimality of A* Graph Search



Optimality of A* Graph Search



Sketch: consider what A* does with a consistent heuristic:

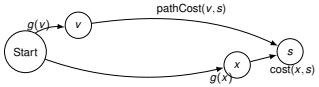
Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)

Fact 2: For every state s, the optimal path is discovered.

Result: A* graph search is optimal

Fact 1 Proof. Previous slide.

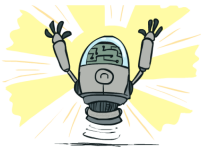
Proof of A* optimality.



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x,s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq pathCost(v,s)$ by induction.
$g(v) + pathCost(v,s) < g(x) + cost(x,s)$
$\implies f(v) < f(s)$.
But then v would have been expanded before s! □

Optimality



Tree search:
- A* is optimal if heuristic is admissible.
- UCS is a special case (h = 0)

Graph search:
- A* optimal if heuristic is consistent
- UCS optimal (h = 0 is consistent)

Consistency implies admissibility

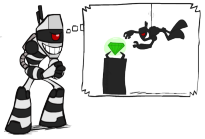
In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

CS 188: Artificial Intelligence

Next:

Constraint Satisfaction Problems.

What is Search For?



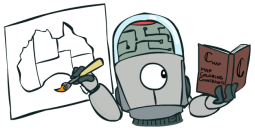
Assumptions about the world:a single agent, deterministic actions, fully observed state, discrete state space.

Planning: sequences of actions
- Want: *path* to the goal.
- Paths have various costs, depths.
- Heuristics give problem-specific guidance

Identification: assignments to variables
- The goal itself is important, not *path*.
- All paths at same depth (for some formulations)

CSPs are specialized identification problems

Constraint Satisfaction Problems



Constraint Satisfaction Problems

Standard search problems:

+ State is a "black box": arbitrary data structure
+ Goal test can be any function over states
+ Successor function can also be anything

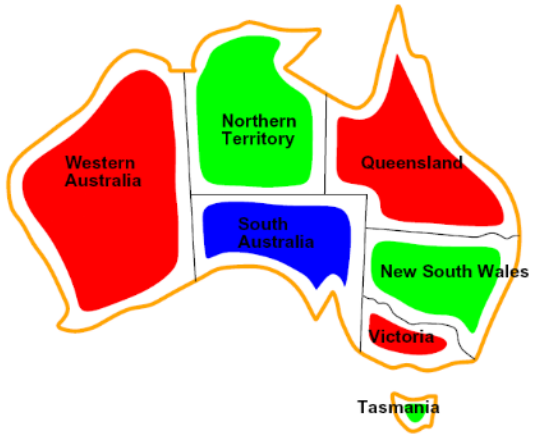
Constraint satisfaction problems (CSPs):

+ A special subset of search problems.
+ State: variables X_i with values from domain D (possibly D_i)
+ Goal test: constraints on legal combinations of values for subsets of variables

Simple example of a formal representation language.

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

CSP Examples



Example: Map Coloring

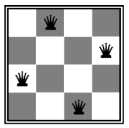
Variables: **WA, NA, Q, NSW, V, SA, T**

Domains: D=red,green,blue

Constraints: adjacent regions must have different colors.
Implicit: WA $\neq$ NT.
Explicit: (WA,NT) $\in$ { **(red,green),(red,blue),...,**}.

Goal Test: do assignments satisfy all constraints?
{ **WA = red, NT= green, Q = red, NSW=green, V=red, SA=blue, T=green** }

Example: N-Queens

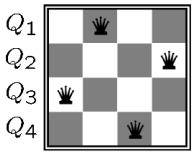


Formulation 1:
Variables: X_{ij}
Domains: $\{0,1\}$

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

$$\sum_{ij} X_{ij} = N$$

Example: N-Queens

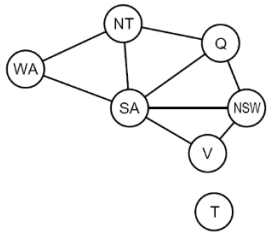


Formulation 2:
Variables: Q_k
Domains: $\{1,2,3,\ldots,N\}$
Constraints:

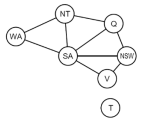
Implicit: $\forall i,j$ non-threatening (Q_i,Q_j)

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

Constraint Graphs



Constraint Graphs



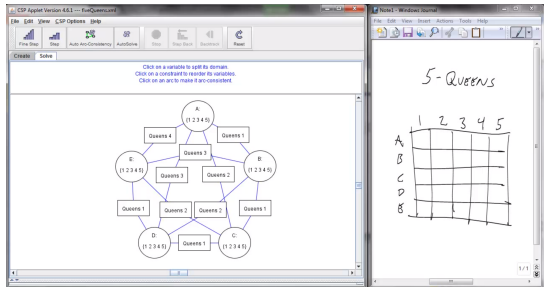
Binary CSP: each constraint relates (at most) two variables

Binary constraint graph: nodes are variables, arcs show constraints

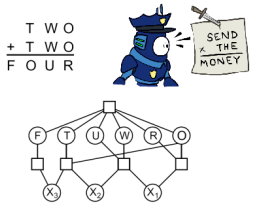
General-purpose CSP algorithms use the graph structure to speed up search.
E.g., Tasmania is an independent subproblem!

[Demo: CSP applet (made available by aispace.org) – n-queens]

5-Queens



Example: Cryptarithmetic



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

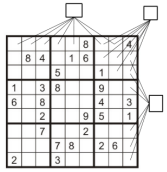
Domains:
$\{0,1,2,3,4,5,6,7,8,9\}$

Constraints:

alldiff(F,T,U,W,R,O).

$O+O=R+10\cdot X_1$
$W+W+X_1=U+10\cdot X_2$
…

Example: Sudoku



Variables:
Each (open) square.

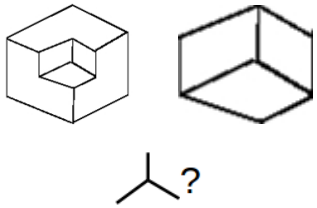
Domains:
$\{1,2,\ldots,9\}$

Constraints:

9-way alldiff for each row
9-way alldiff for each column
9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

Example: The Waltz Algorithm



The Waltz algorithm is for interpreting line drawings of solid polyhedra as 3D objects

An early example of an AI computation posed as a CSP

Approach:
Each intersection is a variable
Adjacent intersections impose constraints on each other
Solutions are physically realizable 3D interpretations

Varieties of CSPs and Constraints

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., Scheduling: Variables = Job start times.
- Linear constraints solvable
- Nonlinear undecidable

Continuous variables
- E.g., start/end times for Hubble Telescope observations
- Linear constraints solvable in polynomial time by LP methods (see cs170 for a bit of this theory)

Varieties of Constraints

Varieties of Constraints.

Unary constraints involve a single variable (equivalent to reducing domains), e.g.:

$$SA \neq \text{green}.$$

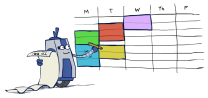
Binary constraints involve pairs of variables, e.g.:

$$SA \neq WA$$

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

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

Real-World CSPs

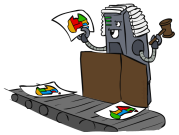


Assignment problems: e.g., who teaches what class
Timetabling problems: e.g., which class is offered when and where?
Hardware configuration
Transportation scheduling
Factory scheduling
Circuit layout
Fault diagnosis
....lots more!

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

Solving CSPs

Standard Search Formulation



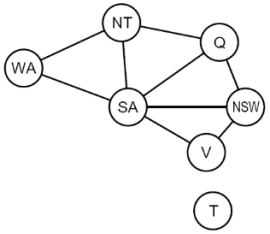
Standard search formulation of CSPs

States defined by the values assigned so far (partial assignments)
- Initial state: the empty assignment,
- Successor function: assign a value to an unassigned variable

Goal test: the current assignment is complete and satisfies all constraints.

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

Search Methods



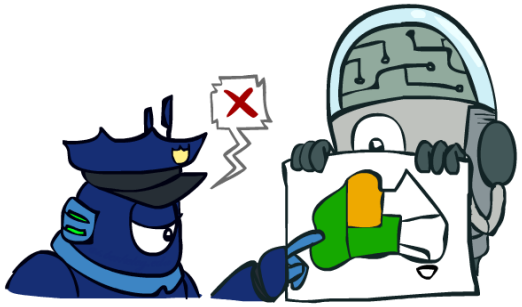
What would BFS do?

What would DFS do?

What problems does naive search have?

Video of Demo Coloring – DFS

Backtracking Search



Backtracking Search

Backtracking search is the basic uninformed algorithm for solving CSPs

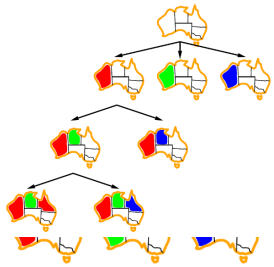
Idea 1: One variable at a time.
Variable assignments are commutative, so fix ordering
I.e., [WA = red then NT = green] same same [NT = green then WA = red]
Assign single variable at each step

Idea 2: Check constraints as you go.
I.e. consider values which do not conflict with 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** (not the best name)

Can solve n-queens for $n \approx 25$

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
What are the choice points?

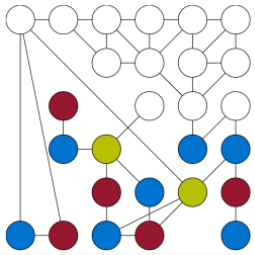
Video of Demo Coloring – Backtracking

CSP-Backtracking Search

CSP-Backtracking = DFS + variable-ordering + fail-on-violation

One optimization possibility: Pick “better” variable orderings and value orderings.

Some issues.



Consider the partially completed CSP assignment.

Decisions made bottom-up, left-to-right. Let X be the decision is obviously doomed in the current assignment.

What is X?

Bonus: How many decisions before CSP-Backtracking search realizes its error?

Improving Backtracking

General-purpose ideas give huge gains in speed

Ordering:
- Which variable should be assigned next?
- In what order should its values be tried?

Filtering:
- Can we detect inevitable failure early?

Structure:
- Can we exploit the problem structure?

Next Time.

Heuristic improvements to CSP search.