LOCAL SEARCH AND CONSTRAINT SATISFACTION

Introduction

- We have already looked in some detail at search techniques.
 - Next lecture we'll go on and look at adversartial search.
- However, there are a couple of other topics we should look at before we get to adversarial search.
 - Local search
 - Constraint satisfaction
 - both of which permeate artificial intelligence.
- They are also useful techniques for all computer scientists to know.

Iterative improvement

- For many problems, the *path* is irrelevant, we just want to find the goal state.
 - Optimization problems
- The state space is the set of configurations.
- We want:
 - the optimum configuration.
 - a configuration that satisfies constraints
- In these cases we can take any state and work to improve it.
 - "Local" since only keep a small part of the state space.
- Constant space.

Travelling salesman

• Problem is to visit all cities once while travelling the shortest distance.



• 13,509 U.S. cities with populations of more than 500 people.



(Rice University, 2003).

cisc3410-fall2010-parsons-lect03a





- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts



• Almost always solves *n*-queens problems almost instantaneously for very large *n*, e.g., *n* ≈ 1 *million*.

Hill-climbing

```
function HILL-CLIMBING(problem) returns a local maximum
inputs: problem, a problem
local variables: current, a node
    neighbor, a node
    current ← MAKE-NODE(INITIAL-STATE[problem])
loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current]
        then return STATE[current]
        current ← neighbor
    end
```

- Hill climbing is also known as:
 - Gradient ascent.
 - Gradient descent.
- Like climbing a hill in the fog with amnesia.
 - All you can do is keep heading up until you get to the top.

• Useful to consider *state space landscape*



10

- *Random-restart hill climbing* overcomes local maxima—trivially complete.
 - Eventually you start from the bottom of every hill.
- *Random sideways moves* escapes from shoulders but loops on flat maxima

Simulated annealing

• Idea: escape local maxima by allowing some "bad" moves

but gradually decrease their size and frequency

• The random jumping around should mean that, over time, we find the highest maximum.

```
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, "temperature"
  current \leftarrow MAKE-NODE(INITIAL-STATE[problem])
  for t \leftarrow 1 to \infty do
      T \leftarrow schedule[t]
      if T = 0 then return current
      next \leftarrow a randomly selected successor of current
      \Delta E \leftarrow \text{VALUE}[next] - \text{VALUE}[current]
      if \Delta E > 0 then current \leftarrow next
      else current \leftarrow next only with probability e^{\Delta E/T}
```

cisc3410-fall2010-parsons-lect03a

• At fixed "temperature" *T*, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

• If *T* is decreased slowly enough we always reach best state *x*^{*} because

$$e^{\frac{E(x^*)}{kT}}/e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$$

for small *T*

• Widely used in VLSI layout, airline scheduling, etc.

Local beam search

- Idea: keep *k* states instead of 1; choose top *k* of all their successors
- Not the same as *k* searches run in parallel!
 - Searches that find good states recruit other searches to join them.
- Problem: quite often, all *k* states end up on same local hill
- Idea: choose *k* successors randomly, biased towards good ones
- Observe the close analogy to natural selection!

Genetic algorithms

• Stochastic local beam search + generate successors from *pairs* of states



- GAs require states encoded as strings (*GPs* use *programs*)
- Crossover helps *iff substrings are meaningful components*



- GAs \neq evolution
 - real genes encode replication machinery!

Constraint satisfaction

- Another approach to optimization.
- In standard search problems a *state* is a "black box"—any old data structure that supports goal test, eval, successor
- In CSP a *state* is defined by *variables* X_i with *values* from a *domain* D_i
- The *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





- Variables: WA, NT, Q, NSW, V, SA, T
- *Domains*: $D_i = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors
 - $-WA \neq NT$, or
 - $\begin{array}{l} \textbf{-} (W\!A, N\!T) \in \\ \{(\textit{red}, \textit{green}), (\textit{red}, \textit{blue}), (\textit{green}, \textit{red}), (\textit{green}, \textit{blue}), \ldots \} \end{array}$

cisc3410-fall2010-parsons-lect03a



Constraint graph

• *Binary CSP*: each constraint relates at most two variables



• *Constraint graph*: nodes are variables, arcs show constraints

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

• Discrete variables

Finite domains; size $d \Rightarrow O(d^n)$ complete assignments

Boolean CSPs, including Boolean satisfiability (NP-complete)
 Infinite domains (integers, strings, etc.)

- 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

- start/end times for Hubble Telescope observations
- linear constraints solvable in polynomial time by LP methods

- *Unary* constraints involve a single variable
 - $-SA \neq green$
- *Binary* constraints involve pairs of variables

 $-SA \neq WA$

- *Higher-order* constraints involve 3 or more variables
 - cryptarithmetic column constraints
- *Preferences* (soft constraints)
 - *red* is better than *green*

often representable by a cost for each variable assignment \rightarrow constrained optimization problems



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), $O + O = R + 10 \cdot X_1$, etc.

Real-world CSPs

- Assignment problems
 - who teaches what class
- Timetabling problems
 - which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning

Notice that many real-world problems involve real-valued variables

Standard search formulation

- Let's start with the straightforward, dumb approach, then fix it
- States are defined by the values assigned so far
 - 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
- This is the same for all CSPs!
- Every solution appears at depth *n* with *n* variables ⇒ use depth-first search
- Path is irrelevant, so can also use complete-state formulation
- $b = (n \ell)d$ at depth ℓ , hence $n!d^n$ leaves!

Backtracking search

- Variable assignments are *commutative*, 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 node so b = d and there are dⁿ leaves.
- Depth-first search for CSPs with single-variable assignments is called *backtracking* search
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve *n*-queens for $n \approx 25$.

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

 $\mathit{var} \leftarrow \mathsf{Select}\text{-}\mathsf{Unassigned}\text{-}\mathsf{Variable}$

(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

• No variables assigned values.



• Assign one variable each of the possible values.



cisc3410-fall2010-parsons-lect03a

• Then take one of those proto-solutions and assign another variable each possible value.



cisc3410-fall2010-parsons-lect03a

• And so on, until you get a solution, or a failure.



- The search has the name *backtracking* because of what happens when the solution fails.
- Search jumps back to the most recent branch point.
 - The "back track"
- Does this method of searching remind you of anything we have seen already?

Improving efficiency

• *General-purpose* methods can give huge gains in speed:

- 1. Which variable should be assigned next?
- 2. In what order should its values be tried?
- 3. Can we detect inevitable failure early?
- 4. Can we take advantage of problem structure?



Degree heuristic

- Tie-breaker among MRV variables
- Choose the variable with the most constraints on remaining variables



• Again, reduces the amount of branching below each choice point.

Least constraining value

- When there are several values to choose from apply this heuristic.
- 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

Forward-checking

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
- This is a form of *inference*.
 - We figure out the effect of the choice of variable value before we get to the relevant point in the search.









Arc-consistency

- Simplest form of propagation makes each arc *consistent*
- $X \rightarrow Y$ is consistent iff for *every* value *x* of *X* there is *some* allowed *y* that *Y* can take.



46







- Arc consistency detects failure earlier than forward checking because of this propagation.
- Run it after each new assignment of values.

function AC-3(csp) returns

the CSP, possibly with reduced domains **inputs**: *csp*, a binary CSP with variables $\{X_1, X_2, ..., X_n\}$ **local variables**: *queue*, a queue of arcs, initially all the arcs in *csp*

```
while queue is not empty do

(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue)

if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then

for each X_k in NEIGHBORS[X_i] do

add (X_k, X_i) to queue
```

function REMOVE-INCONSISTENT-VALUES(X_i , X_j) **returns** true iff succeeds *removed* \leftarrow *false* **for each** x **in** DOMAIN[X_i] **do if** no value y **in** DOMAIN[X_j] allows (x,y) to satisfy the constraint $X_i \leftrightarrow X_j$ **then** delete x from DOMAIN[X_i]; *removed* \leftarrow *true* **return** *removed*

Summary

- We have looked at some variations of search that work when we are only interested in the solution, *not* the path.
- We looked at local search:
 - Iterative improvement
 - Hill-climbing
 - Simulated annealing
 - Genetic algorithms
- Then we looked at constraint propagation.
- We only scratched the surface of all of these topics the textbook covers much more on both topics.