Beyond Classical Search

Professor Marie Roch

Chapter 4, Russell & Norvig

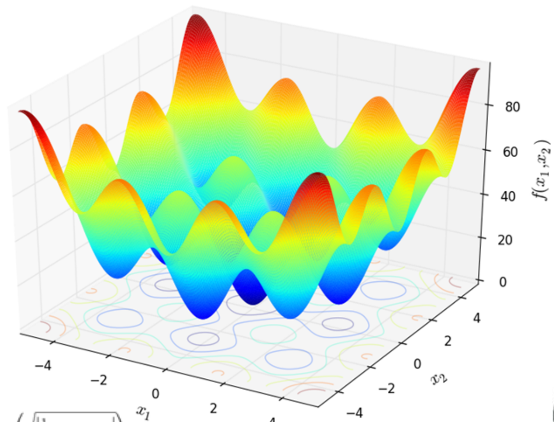
Local search

- Single state node
  - paths not usually retained
  - typically move only to neighbors of state
- The good
  - Low memory usage
  - Appropriate for large (possibly infinite) state spaces
- The bad
  - Lose advantages from search-tree retention (e.g. backtracking)

2

Optimization problems

- Find best state – find extrema of objective function f(state)



$$f_{\text{EggHolder}}(\mathbf{x}) = -x_1 \sin\left(\sqrt{|x_1 - x_2 - 47|}\right) - (x_2 + 47)\sin\left(\sqrt{\left|\frac{1}{2}x_1 + x_2 + 47\right|}\right)$$

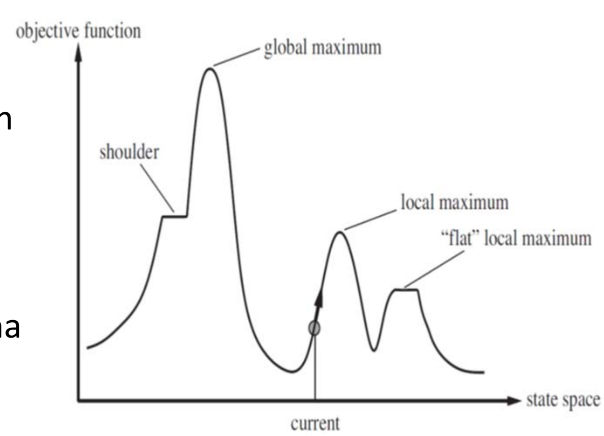
infinity77.net

Optimization problems

- Optimal solutions (global extrema) can be problematic

- Complete search
  →local extrema
  easy to get stuck

- Optimal search
  →global extrema



Hill-climbing (aka greedy) search

```
def hillclimb(state):
  done = False
  while not done:
    next = successors of state
    find s in next such that maximizes f(s) – f(state)
    if f(s) – f(state) > 0 then state = s
      else done = True
  return state
```


Troublesome for hill climbing…

- Local extrema – trapped!
- Ridges – no real way out
- Plateaus – what should we do for sideways moves? Continue?

Hill-climbing variants

- Stochastic – Assign probabilities related to steepness of choice and pick randomly (slow convergence).
- First-choice – Generate successors randomly, pick the first one that's better than current state.
- Random-restart – Pick a new initial state if we don't find what we are looking for.

Speed at which search converges to a "good" state?

7

Simulated annealing

- Annealing
  - Process to harden metals
  - Subject to high heat
    - metals enter high energy state
    - slowly cool
    - allows molecules to realign, reducing stress
- Simulated annealing
  - Simulate temperature
  - Volatility of action choices is related to temperature
    - high temperature – more likely to pick "risky" decisions
    - low temperature – more likely to pick "good" decisions

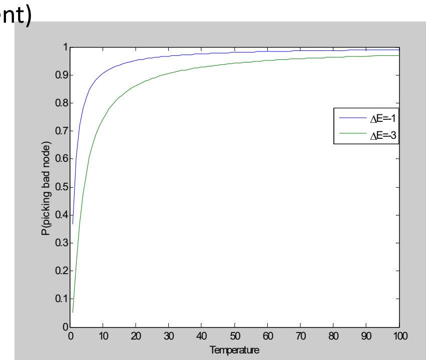
8

Simulated annealing

- Simulated annealing
  - "temperature" starts hot and cools (function of time)
  - A successor state is chosen at random
    - improvement + or degradation - of state fitness ΔE=fitness(child)-fitness(current)
    - If ΔE >0 then update state otherwise

      update based on odds of picking a bad node

      $$1 + e^{\Delta E/Temp}$$



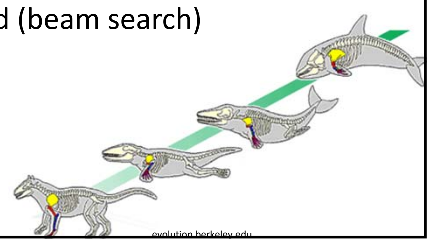
Beam search

- Differs in treatment of successors from standard search
  - Only keep the k most successful children
  - May add stochastic component to increase diversity of population
- Frequently used to explore multiple hypotheses while keeping frontier set small
- Example: Speech recognition systems often use this

10

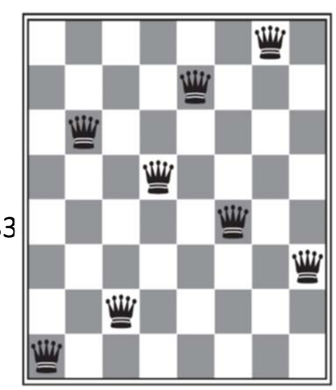
Genetic algorithms

- Search-state nodes are measured by a fitness function
- Successors
  - Generated from random pair in frontier (called population)
    - new state from crossover (mixture of parent states)
    - new state may be further mutated
  - Only fittest nodes are retained (beam search)



Genetic algorithms

- States need to be represented in a way that parameters can be mixed
- Example
  - 8 queens with all queens placed
  - state – row # of queen (1,6,2,5,7,4,8,3) or 16257483
  - fitness function: # non-attacking pairs



Genetic algorithm example

- How are random pairs selected?

  Assigned probabilities

  $$P(\text{node}) = \frac{fitness(node)}{\sum_{i \in population} fitness(i)}$$

- Population of four nodes

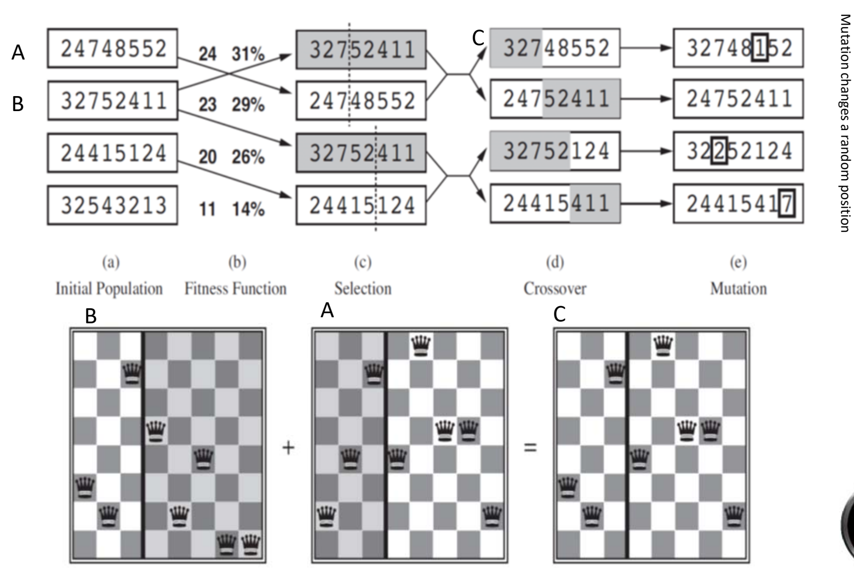
  fitness(24748552)= 24 → 31% (24/(24+23+20+11))

  fitness(32752411)= 23 → 29%

  fitness(24415124)= 20 → 26%

  fitness(32543213)= 11 → 14%

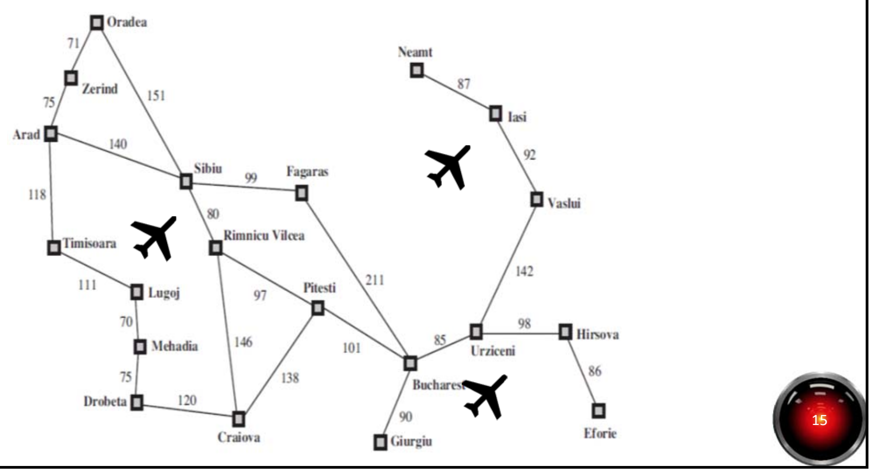
Genetic algorithm example



Local search in continuous spaces

Place three airports to minimize distance to nearest city

Abstract view of Romanian roads
(Russel and Norvig 2010, Fig 3.2)



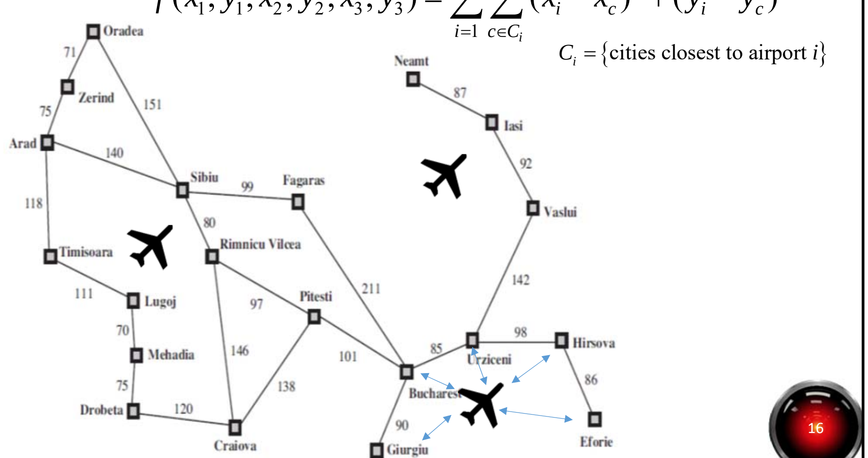
Local search in continuous spaces

Place three airports to minimize distance to nearest city

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

$C_i = \{\text{cities closest to airport } i\}$

Abstract view of Romanian roads
(Russel and Norvig 2010, Fig 3.2)



Local search in continuous spaces

Possible approaches

- Discretize the search space
  - increment state by $\pm\varepsilon$
  - with 6 variables, 12 possible successors (if constrained to one direction)
  - what size ε?

- Compute the gradient
  - Gives us the direction of steepest ascent.

$$\nabla f = \left(\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta y_1}, \frac{\delta f}{\delta x_2}, \frac{\delta f}{\delta y_2}, \frac{\delta f}{\delta x_3}, \frac{\delta f}{\delta y_3} \right)$$

Local search in continuous space

Gradient approaches

- If gradient exists in closed form, may be able to solve for maximum: $\nabla f = 0$

- Many objective functions cannot be solved in closed form, e.g.

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

  has discontinuities as cities change C_i membership.

Local search in continuous space

- Local gradient might be possible

$$\nabla f(x_1, y_1, x_2, y_2, x_3, y_3) = \left(2\sum_{c \in C_1}(x_1 - x_c), 2\sum_{c \in C_1}(y_1 - y_c), \ldots \right)$$

- If objective function not differentiable evaluate f in the neighborhood and compute *empirical gradient*.

- Update requires step size α

$$state \leftarrow state + \alpha \nabla f(state)$$

19

Local search in continuous space

- Choice of α
  - too small… learning slow
  - too large… might overshoot extrema or gradient change

- Line search
  - double α repeatedly until objective function f starts to decrease
  - choose new direction

20

Newton-Raphson method

- Method for finding roots $f(x) = 0$
- Find root x:
  - start with a "good" estimate x_0
  - improve it iteratively
- Suppose we pick $x_0=a$ and actual root is r; $f(r)=0$
- Let $a + h = r$

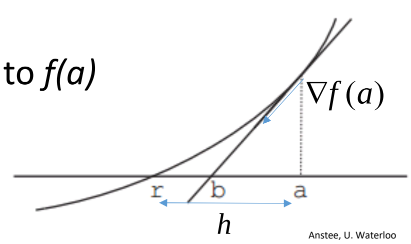
21

Newton-Raphson method

- So, we have

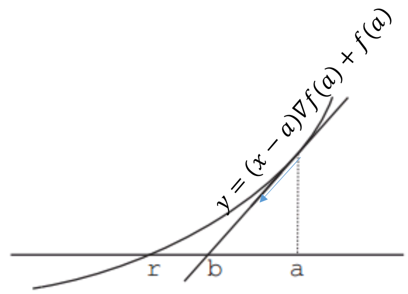
$$f(r) = 0, x_0 = a \text{ and let } r = a + h$$
$$f(r) = f(a + h)$$

- Consider the line tangent to *f(a)* given by $\nabla f(a)$.
- It intercepts the x axis at b



22

Newton-Raphson method



Anstee, U. Waterloo

tangent line through (b,0) and (a,f(a)): $y=(x-a)\nabla f(a)+f(a)$

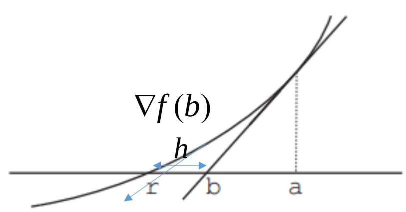
Let's find b's value by setting y=0

$0=(x-a)\nabla f(a)+f(a)\rightarrow$

$$x=a-\frac{f(a)}{\nabla f(a)}$$

Newton-Raphson method

- Linear approximation $x_{i+1}=a-\frac{f(a)}{\nabla f(a)}$ provides a new approximation of the root.



Anstee, U. Waterloo

- Iterate until convergence
- Very good with good starting points, not so good with bad ones…

Newton-Raphson and local search

- We want to find states where gradient of optimization function is zero: $\nabla f(x) = 0$
- Newton-Raphson lets us find this, but we use the derivative of the gradient, or second derivative

25

Newton-Raphson method

- In airport optimization, we computed $\partial f / \partial x_i$ and $\partial f / \partial y_i$
- As we find the roots of the derivative, we need to find $\partial^2 f / \partial x_i \partial x_j$ and $\partial^2 f / \partial y_i \partial y_j$ and $\partial^2 f / \partial x_i \partial y_j$

$$\partial^2 f / \partial x_1 \partial y_2 \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

$$= \partial f / \partial y_2 \left(2 \sum_{c \in C_1} (x_1 - x_c) \right) \partial x_1$$

$$= 0$$

$$\partial^2 f / \partial x_1 \partial x_1 \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

$$= \partial f / \partial x_1 \left(2 \sum_{c \in C_1} (x_1 - x_c) \right) \partial x_1$$

$$= 2$$

26

Newton-Raphson method

- Derivatives can be arranged in Hessian matrix

$$H_f(x) = \begin{bmatrix} {\partial^2 f}/{\partial x_1 \partial x_1} & {\partial^2 f}/{\partial x_1 \partial x_2} & {\partial^2 f}/{\partial x_3 \partial x_1} & \cdots & & {\partial^2 f}/{\partial xy_3 \partial y_3} \\ {\partial^2 f}/{\partial x_2 \partial x_1} & {\partial^2 f}/{\partial x_2 \partial x_2} & {\partial^2 f}/{\partial x_3 \partial x_2} & \cdots & & \\ & & {\partial^2 f}/{\partial x_3 \partial x_3} & \cdots & & \\ \vdots & \vdots & {\partial^2 f}/{\partial x_3 \partial y_1} & {\partial^2 f}/{\partial y_1 \partial y_1} & \vdots & \vdots \\ & & {\partial^2 f}/{\partial x_3 \partial y_2} & \cdots & {\partial^2 f}/{\partial y_2 \partial y_2} & \\ {\partial^2 f}/{\partial xy_3 \partial y_3} & & {\partial^2 f}/{\partial x_3 \partial y_3} & \cdots & & {\partial^2 f}/{\partial y_2 \partial y_2} \end{bmatrix}$$

In this case diagonals are 2,
off diagonals are 0

$$\begin{bmatrix} 2 & & & & & \\ & 2 & & & & \\ & & 2 & & & \\ & & & 2 & & \\ & & & & 2 & \\ & & & & & 2 \end{bmatrix}$$

27

Newton-Raphson method

Update function becomes

$$x_{i+1} = x - H_f^{-1}(x_i) \nabla f(x_i)$$

where $H_f^{-1}(x_i)$ is the inverse of the Hessian matrix $H_f(x_i)$

We will not cover *constrained optimization*
which lets us add conditions that must hold, e.g.:

(x_i, y_i) cannot be on a mountain
(x_i, y_i) cannot be in a lake

28

Actions and contingency plans

- Deterministic
  - Percepts only needed for initial state
  - We know the results of every action
- Non-deterministic
  - No longer sure what the next state is
- Partially observable
  - Might not be certain of initial state

Non-deterministic/partially observable environments require *contingency plans* (aka strategies)

29

Contingency plans

- We redefine the result of an action such that it returns multiple possible states.

Example for a partially observable environment

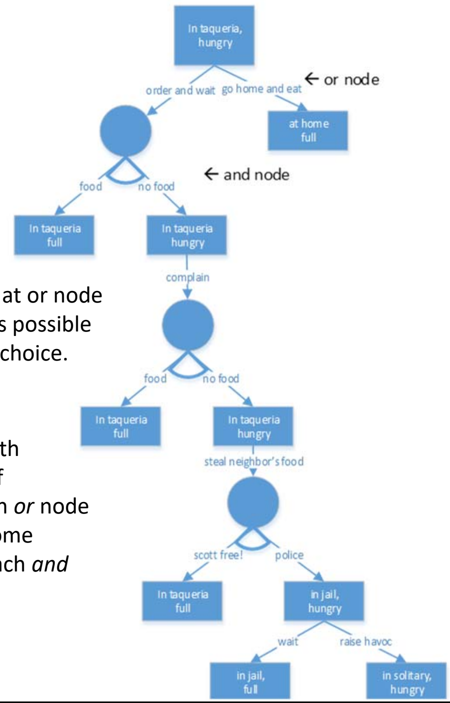
result(state(xy(32,45), ok), deltaxy_m(0,3))←
{ state(xy(32,48), falling), state(xy(32,48), ok) }

See erratic vacuum world section 4.3.1 for a more developed example

30

And-or search trees

allow representation of multiple outcomes



Agent makes choice at or node
And node represents possible outcomes of that choice.

Solution: **subtree** with
- Goal at every leaf
- 1+ action for each *or* node
- includes all outcome branches from each *and* node

31

And-or search

- To simplify, assume a single start state
- Expand the node and take actions
  - or nodes – represent deterministic choices
  - and nodes – environment decides outcome of an action (nondeterministic as far as agent is concerned)
- With or nodes, we continue searching for a solution.
- With and nodes, there needs to be a solution along every node of the and.

32

And-or search

function AND-OR-GRAPH-SEARCH(*problem*) **returns** *a conditional plan, or failure*
 OR-SEARCH(*problem*.INITIAL-STATE, *problem*, [ ])

---

function OR-SEARCH(*state*, *problem*, *path*) **returns** *a conditional plan, or failure*
 if *problem*.GOAL-TEST(*state*) **then return** the empty plan
 if *state* is on *path* **then return** *failure*
 for each *action* **in** *problem*.ACTIONS(*state*) **do**
  plan ← AND-SEARCH(RESULTS(*state*, *action*), *problem*, [*state* | *path*])
  if *plan* ≠ *failure* **then return** [*action* | *plan*]
 return *failure*

---

function AND-SEARCH(*states*, *problem*, *path*) **returns** *a conditional plan, or failure*
 for each s_i **in** *states* **do**
  $plan_i$ ← OR-SEARCH(s_i, *problem*, *path*)
  if $plan_i$ = *failure* **then return** *failure*
 return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** ... **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

Figure 4.11, R&N

38