Title: Informed Search Methods
Required reading: AIMA, Chapter 4 (Sections 4.1, 4.2, & 4.3)
LWH: Chapters 6, 10, 13 and 14.

Introduction to Artificial Intelligence
CSCE 476-876, Spring 2009
URL: `www.cse.unl.edu/~choueiry/S09-476-876`

Berthe Y. Choueiry (Shu-we-ri)
choueiry@cse.unl.edu, (402)472-5444

Outline

- Categorization of search techniques
- Ordered search (search with an evaluation function)
- Best-first search:
  (1) Greedy search (2) A*
- Admissible heuristic functions:
  how to compare them?
  how to generate them?
  how to combine them?
- Iterative improvement search:
  (1) Hill-climbing (2) Simulated annealing

Types of Search (I)

1- Uninformed vs. informed

2- Systematic/constructive vs. iterative improvement

Uninformed :

use only information available in problem definition,

no idea about distance to goal

→ can be incredibly ineffective in practice

Heuristic :

exploits some knowledge of the domain

also useful for solving optimization problems

Types of Search (II)

Systematic, exhaustive, constructive search:

a partial solution is incrementally extended into global solution

Partial solution =

sequence of transitions between states

Global solution =

Solution from the initial state to the goal state

Examples: { Uninformed
Informed (heuristic): Greedy search, A^*

→ Returns the path; solution = path

Types of Search (III)

Iterative improvement:

A state is gradually modified and evaluated until reaching an (acceptable) optimum

→ We don't care about the path, we care about 'quality' of state
→ Returns a state; a solution = good quality state
→ Necessarily an informed search

Examples (informed):
- Hill climbing
- Simulated Annealing (physics), Taboo search
- Genetic algorithms (biology)

Ordered search

• Strategies for systematic search are generated by choosing which node from the fringe to expand first

• The node to expand is chosen by an **evaluation function**, expressing 'desirability' ⟶ **ordered search**

• When nodes in queue are sorted according to their decreasing values by the evaluation function ⟶ **best-first search**

• Warning: 'best' is actually 'seemingly-best' given the evaluation function. Not always best (otherwise, we could march directly to the goal!)

Search using an evaluation function

- Example: uniform-cost search!

  What is the evaluation function?

  Evaluates cost from .... to ....?

- How about the cost **to** the goal?

  $h(n)$ = estimated cost of the cheapest
  path from the state at node n to a goal state

  $h(n)$ would help focusing search

Cost to the goal

This information is not part of the problem description

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Best-first search

1. Greedy search chooses the node n closest to the goal such as $h(n)$ is minimal

2. A* search chooses the least-cost solution

$$\text{solution cost } f(n) \begin{cases} g(n)\text{: cost from root to a given node } n \\ + \\ h(n)\text{: cost from the node } n \text{ to the goal node} \end{cases}$$

such as $f(n) = g(n) + h(n)$ is minimal

Greedy search

→ First expand the node whose state is 'closest' to the goal!

→ Minimize $h(n)$

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence
  inputs: problem, a problem
          Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(problem, Queueing-Fn)
```

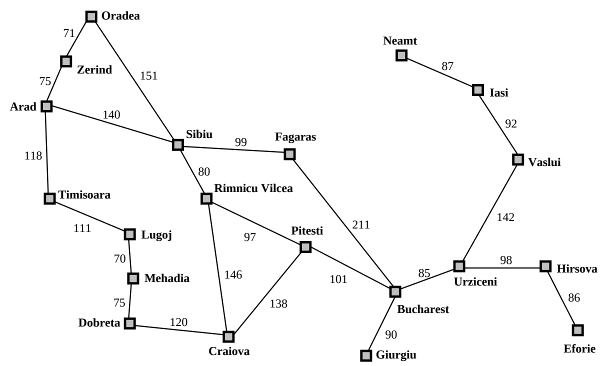
→ Usually, cost of reaching a goal may be estimated, not determined exactly

→ If state at n is goal, $h(n)$= ?

→ How to choose $h(n)$? Problem specific! Heuristic!

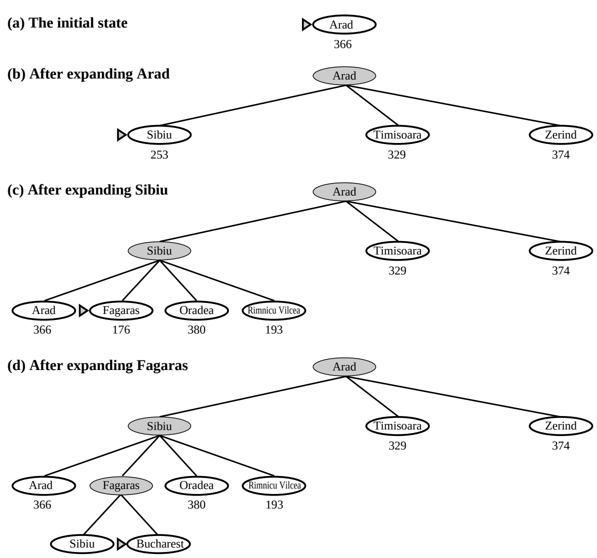
Greedy search: Romania

$h_{\mathtt{SLD}}(n)$ = straight-line distance between n and goal location



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

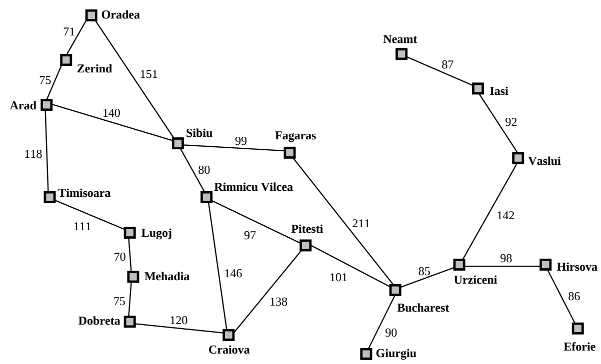
Greedy search: Trip from Arad to Bucharest



... Greedy search! quick, but not optimal!

Greedy search: Problems

From Iasi to Fagaras? { False starts: Neamt is a dead-end; Looping }



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy search: Properties

→ Like depth-first, tends to follow a single path to the goal

→ Like depth-first { Not complete; Not optimal }

→ Time complexity: $O(b^m)$, m maximum depth

→ Space complexity: $O(b^m)$ retains all nodes in memory

→ Good h function (considerably) reduces space and time
but h functions are problem dependent :—(

Hmm...

Greedy search minimizes estimated cost to goal $h(n)$
→ cuts search cost considerably
→ but not optimal, not complete

Uniform-cost search minimizes cost of the path so far $g(n)$
→ is optimal and complete
→ but can be wasteful of resources

New-Best-First search minimizes $f(n) = g(n) + h(n)$
→ combines greedy and uniform-cost searches
$f(n)$ = estimated cost of cheapest solution via n
→ Provably: complete and optimal, if $h(n)$ is admissible

A* Search

- **A* search**
  Best-first search expanding the node in the fringe with minimal $f(n) = g(n) + h(n)$
- **A* search with admissible** $h(n)$
  Provably complete, optimal, and optimally efficient using TREE-SEARCH
- **A* search with consistent** $h(n)$
  Remains optimal even using GRAPH-SEARCH

(See TREE-SEARCH page 72 and GRAPH-SEARCH page 83)

Admissible heuristic

An admissible heuristic is a heuristic that <u>never overestimates</u> the cost to reach the goal

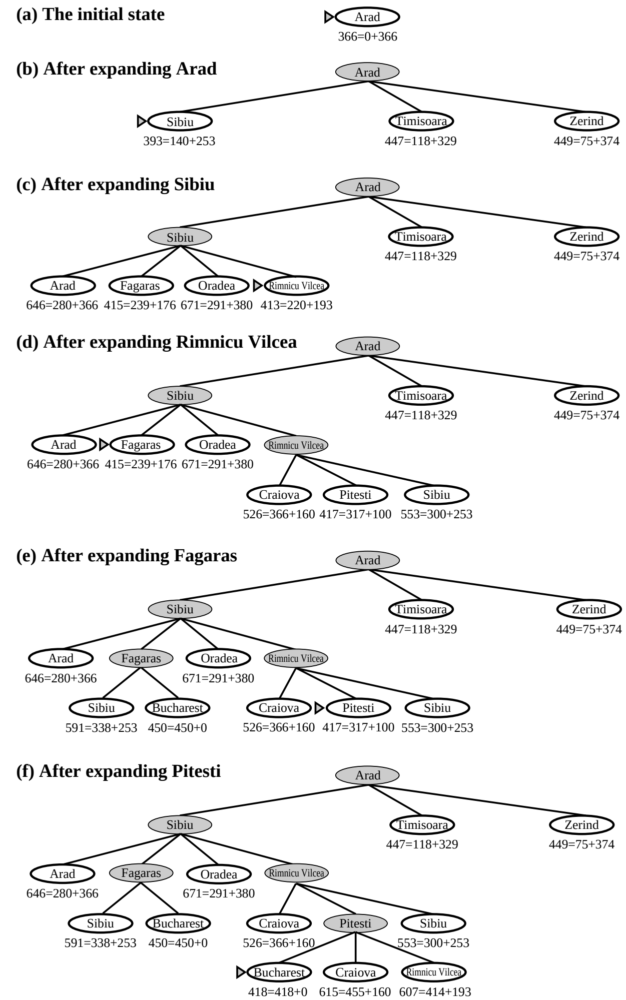
→ is optimistic

→ thinks the cost of solving is less than it actually is

Example: { travel: straight line distance; I can finish college in 3 years; We can fly to Mars by 2003 }

If h is admissible,
$f(n)$ never overestimates the actual cost of the best solution through n.

A* Search From Arad to Bucharest



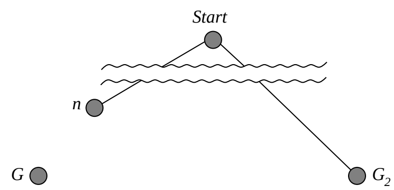
A* Search is optimal

G, G_2 goal states $\Rightarrow g(G) = f(G),\ f(G_2) = g(G_2)$ $\qquad h(G) = h(G_2) = 0$

G optimal goal state $\Rightarrow C^* = f(G)$

G_2 suboptimal $\Rightarrow f(G_2) > C^* = f(G)$ (1)

Suppose n is not chosen for expansion



h admissible $\Rightarrow C^* \geq f(n)$ (2)

Since n was not chosen for expansion $\Rightarrow f(n) \geq f(G_2)$ (3)

(2) + (3) $\Rightarrow C^* \geq f(G_2)$ (4)

(1) and (4) are contradictory $\Rightarrow n$ should be chosen for expansion

Which nodes does A* expand?

Goal-Test is applied to State(node) when a node is chosen from the fringe for expansion, not when the node is generated

Theorem 3 & 4 in Pearl 84, original results by Nilsson

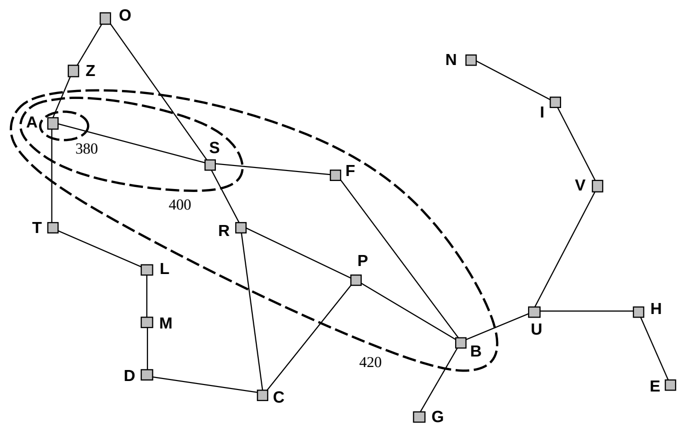
- *Necessary condition:* Any node expanded by A* cannot have an f value exceeding C^*: For all nodes expanded, $f(n) \leq C^*$
- *Sufficient condition:* Every node in the fringe for $f(n) < C^*$ will eventually be expanded by A*

In summary

- A* expands all nodes with $f(n) < C^*$
- A* expands some nodes with $f(n) = C^*$
- A* expands no nodes with $f(n) > C^*$

Expanding contours

A* expands nodes from fringe in increasing f value
We can conceptually draw contours in the search space



The first solution found is necessarily the optimal solution
Careful: a TEST-GOAL is applied at node expansion

A* Search is complete

Since A* search expands all nodes with $f(n) < C^*$, it must eventually reach the goal state unless there are infinitely many nodes $f(n) < C^*$:

- 1. ∃ a node with infinite branching factor
- or
- 2. ∃ a path with infinite number of nodes along it

A* is complete if:

- on locally finite graphs
- and
- $\exists \delta > 0$ constant, the cost of each operator $> \delta$

A^* Search Complexity

Time:
Exponential in (relative error in $h \times$ length of solution path)
... quite bad

Space: must keep all nodes in memory
Number of nodes within goal contour is exponential in length of solution.... unless the error in the heuristic function $|h(n) - h^*(n)|$ grows no faster than the log of the actual path cost: $|h(n) - h^*(n)| \leq O(\log h^*(n))$
In practice, the error is proportional... impractical..

major drawback of A^*: runs out of space quickly

$\rightarrow$ Memory Bounded Search IDA*(not addressed here)

A^* Search is optimally efficient

.. for any given evaluation function: no other algorithms that finds the optimal solution is guaranteed to expend fewer nodes than A^*

Interpretation (proof not presented): Any algorithm that does not expand all nodes between root and the goal contour risks missing the optimal solution

Tree-Search vs. Graph-Search

After choosing a node from the fringe and before expanding it, Graph-Search checks whether State(node) was visited before to avoid loops.

→ Graph-search may lose optimal solution

Solutions

1. In Graph-Search, discard the more expensive path to a node
2. Ensure that the optimal path to any repeated state is the first one found
   → Consistency

Consistency

$h(n)$ is consistent
If $\forall$ n and $\forall$ n' successor of n along a path, we have
$h(n) \leq k(n, n') + h(n')$, k cost of cheapest path from n to n'

Monotonicity

$h(n)$ is monotone
If $\forall$ n and $\forall$ n' successor of n generated by action a, we have
$h(n) \leq c(n, a, n') + h(n')$, n' is an <u>immediate</u> successor of n
Triangle inequality ($\langle n, n'$, goal$\rangle$)

Important: h is consistent $\Leftrightarrow$ h is monotone

Beware: of confusing terminology 'consistent' and 'monotone'
Values of h not necessarily decreasing/nonincreasing

Properties of h: Important results

- h consistent $\Leftrightarrow$ h monotone (Pearl 84)
- h consistent $\Rightarrow$ h admissible (AIMA, Exercise 4.7)
  consistency is stricter than admissibility
- h consistent $\Rightarrow$ f is nondecreasing
  $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$
- h consistent $\Rightarrow$ A* using GRAPH-SEARCH is optimally efficient

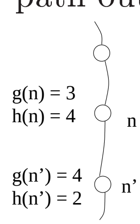
Pathmax equation

You may ignore this slide

Monotonicity of f: values along a path are nondecreasing
When f is not monotonic, use **pathmax** equation

$$f(n') = max(f(n), g(n') + h(n'))$$

A* never decreases along any path out from root



Pathmax
- guarantees f nondecreasing
- does not guarantee h consistent
- does not guarantee A* + GRAPH-SEARCH is optimally efficient

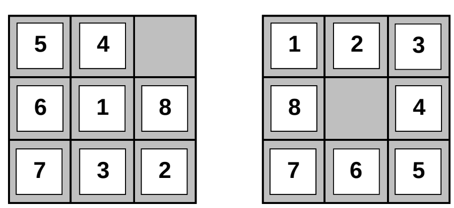
Summarizing definitions for A*

- A* is a best-first search that expands the node in the fringe with minimal $f(n) = g(n) + h(n)$
- An admissible function h never overestimates the distance to the goal.
- h admissible $\Rightarrow$ A* is complete, optimal, optimally efficient using Tree-Search
- h consistent $\Leftrightarrow$ h monotone
  h consistent $\Rightarrow$ h admissible
  h consistent $\Rightarrow$ f nondecreasing
- h consistent $\Rightarrow$ A* remains optimal using Graph-Search

Admissible heuristic functions

Examples

- Route-finding problems: straight-line distance
- 8-puzzle: $\begin{cases} h_1(n) = \text{number of misplaced tiles} \\ h_2(n) = \text{total Manhattan distance} \end{cases}$



Start State Goal State

$h_1(S) = ?$
$h_2(S) = ?$

Performance of admissible heuristic functions

Two criteria to compare admissible heuristic functions:

1. Effective branching factor: b^*
2. Dominance: number of nodes expanded

Effective branching factor b^*

– The heuristic expands N nodes in total

– The solution depth is d

$\longrightarrow$ b^* is the branching factor had the tree been uniform

$$N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d = \frac{(b^*)^{d+1} - 1}{b^* - 1}$$

– Example: N=52, d=5 $\rightarrow$ $b^* = 1.92$

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 *dominates* h_1 and is better for search

Typical search costs: nodes expanded

Sol. depth	**IDS**	$\mathbf{A}^*(h_1)$	$\mathbf{A}^*(h_2)$
$d = 12$	3,644,035	227	73
$d = 24$	too many	39,135	1,641

A* expands all nodes $f(n) < C^* \Rightarrow g(n) + h(n) < C^*$
$\Rightarrow h(n) < C^* - g(n)$

If $h_1 \leq h_2$, A* with h_1 will always expand at least as many (if not more) nodes than A* with h_2

$\longrightarrow$ It is always better to use a heuristic function with higher values, as long as it does not overestimate (remains admissible)

How to generate admissible heuristics?

$\rightarrow$ Use *exact* solution cost of a relaxed (easier) problem

Steps:
- Consider problem P
- Take a problem P' easier than P
- Find solution to P'
- Use solution of P' as a heuristic for P

Relaxing the 8-puzzle problem

A tile can move mode square A to square B if
A is (horizontally or vertically) adjacent to B and B is blank

1. A tile can move from square A to square B if A is adjacent to B
   The rules are relaxed so that a tile can move to *any adjacent square*: the shortest solution can be used as a heuristic ($\equiv h_2(n)$)
2. A tile can move from square A to square B if B is blank
   Gaschnig heuristic (Exercice 4.9, AIMA, page 135)
3. A tile can move from square A to square B
   The rules of the 8-puzzle are relaxed so that a tile can move *anywhere*: the shortest solution can be used as a heuristic ($\equiv h_1(n)$)

An admissible heuristic for the TSP

Let path be *any* structure that connects all cities
$\Longrightarrow$ minimum spanning tree heuristic (polynomial)

(Exercice 4.8, AIMA, page 135)

Combining several admissible heuristic functions

We have a set of admissible heuristics $h_1, h_2, h_3, \ldots, h_m$ but no heuristic that dominates all others, what to do?

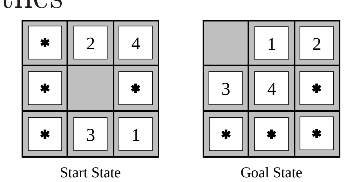
$$\longrightarrow h(n) = \texttt{max}(h_1(n), h_2(n), \ldots, h_m(n))$$

h is admissible and dominates all others.

→ Problem:
Cost of computing the heuristic (vs. cost of expanding nodes)

Using subproblems to derive an admissible heuristic function

Goal: get 1, 2, 3, 4 into their correct positions, ignoring the 'identity' of the other tiles



Cost of optimal solution to subproblem used as a lower bound (and is substantially more accurate than Manhattan distance)

Pattern databases:

- Identify patterns (which represent several possible states)
- Store cost of <u>exact</u> solutions of patterns
- During search, retrieve cost of pattern and use as a (tight) estimate

Cost of building the database is amortized over 'time'

Iterative improvement (a.k.a. local search)

⟶ Sometimes, the 'path' to the goal is irrelevant
only the state description (or its quality) is needed

Iterative improvement search

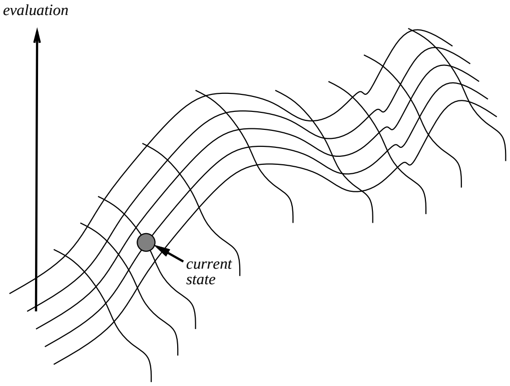
- choose a single current state, sub-optimal
- gradually modify current state
- generally visiting 'neighbors'
- until reaching a near-optimal state

Example: complete-state formulation of N-queens

Main advantages of local search techniques

1. Memory (usually a constant amount)
2. Find reasonable solutions in large spaces
   where we cannot possibly search the space exhaustively
3. Useful for optimization problems:
   best state given an objective function (quality of the goal)

Intuition: state-scape landscape



- All states are layed up on the surface of a landscape
- A state's location determines its neighbors (where it can move)
- A state's elevation represents its quality (value of objective function)
- Move from one neighbor of the current state to another state until reaching the highest peak

Two major classes

1. Hill climbing (a.k.a. gradient ascent/descent)
   → try to make changes to improve quality of current state
2. Simulated Annealing (physics)
   → things can <u>temporarily</u> get worse

Others: tabu search, local beam search, genetic algorithms, etc.

⟶ Optimality (soundness)? Completeness?
⟶ Complexity: space? time?

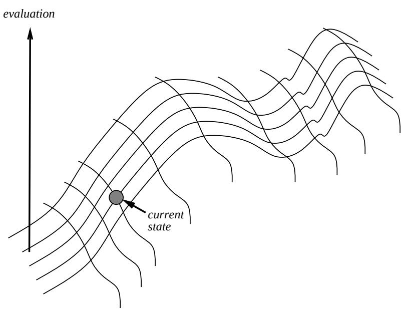
⟶ In practice, surprisingly good.. (eroding myth)

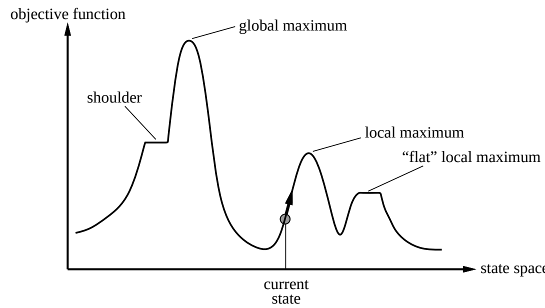
Hill climbing

Start from any state at random and loop:

Examine all direct neighbors

If a neighbor has higher value then move to it else exit

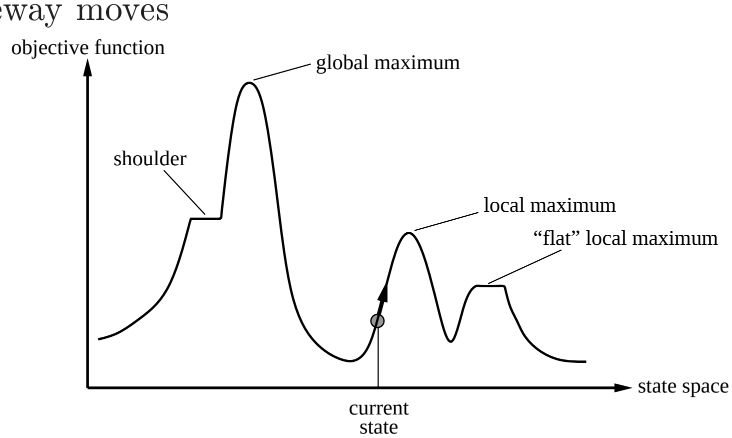




Problems:
- Local optima: (maxima or minima) search halts
- Plateau: flat local optimum or shoulder
- Ridge

Plateaux

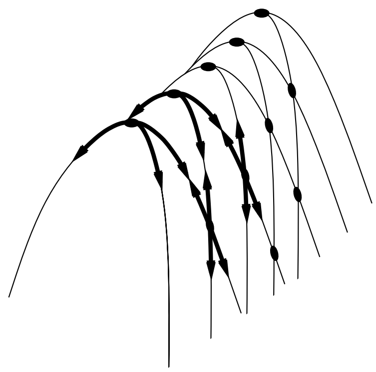
Allow sideway moves



- For shoulder, good solution
- For flat local optima, may result in an infinite loop
  Limit number of moves

Ridges

Sequence of local optima that is difficult to navigate



Variants of Hill Climbing

- Stochastic hill climbing: random walk
  Choose to disobey the heuristic, sometimes
  Parameter: How often?
- First-choice hill climbing
  Choose first best neighbor examined
  Good solution when we have too many neighbors
- Random-restart hill climbing
  A series of hill-climbing searches from random initial states

Random-restart hill-climbing

→ When HC halts or no progress is made
re-start from a different (randomly chosen) starting
save best results found so far

→ Repeat random restart
- for a fixed number of iterations, or
- until best results have not been improved for a certain number of iterations

Simulated annealing (I)

Basic idea: When stuck in a local maximum allow few steps towards less good neighbors to escape the local maximum

Start from any state at random, start count down and loop until time is over:

Pick up a neighbor at <u>random</u>
Set ΔE = value(neighbor) - value(current state)
If ΔE>0 (neighbor is better)
then move to neighbor
else ΔE<0 move to it with probability < 1

Transition probability $\simeq e^{\Delta E/T}$ $\begin{cases} \Delta\text{E is negative} \\ \text{T: count-down time} \end{cases}$

as time passes, less and less likely to make the move towards 'unattractive' neighbors

Simulated annealing (II)

Analogy to physics:

Gradually cooling a liquid until it freezes
If temperature is lowered sufficiently slowly, material will attain lowest-energy configuration (perfect order)

Count down	$\longleftrightarrow$	Temperature
Moves between states	$\longleftrightarrow$	Thermal noise
Global optimum	$\longleftrightarrow$	Lowest-energy configuration

How about decision problems?

Optimization problems		**Decision problems**
Iterative improvement	$\longleftrightarrow$	Iterative repair
State value	$\longleftrightarrow$	Number of constraints violated
Sub-optimal state	$\longleftrightarrow$	Inconsistent state
Optimal state	$\longleftrightarrow$	Consistent state

Local beam search

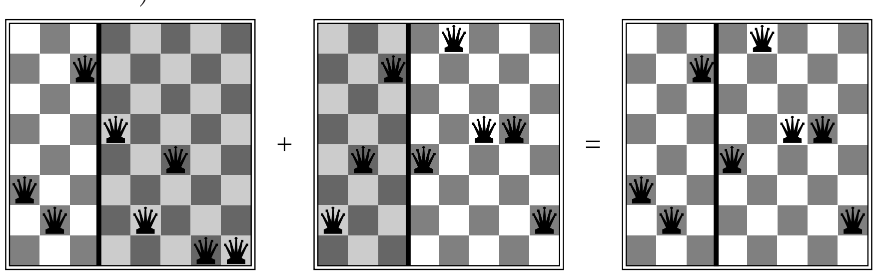
- Keeps track of k states
- Mechanism:
  Begins with k states
  At each step, all successors of all k states generated
  Goal reached? Stop.
  Otherwise, selects k best successors, and repeat.
- Not exactly a k restarts: k runs are not independent
- Stochastic beam search increases diversity

B.Y. Choueiry

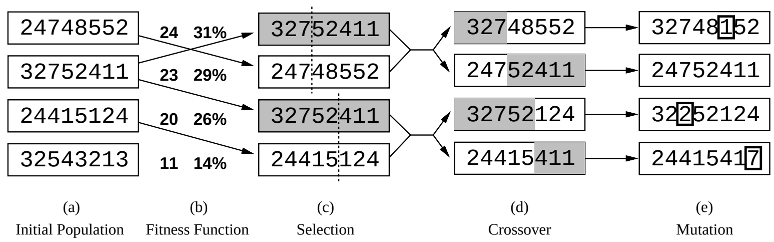
Instructor's notes #7
February 2, 2009

Genetic algorithms

- Basic concept: combines two (parent) states
- Mechanism:
  Starts with k random states (population)
  Encodes individuals in a compact representation (e.g., a string in an alphabet)
  Combines partial solutions to generate new solutions (next generation)



Important components of a genetic algorithm



- Fitness function ranks a state's quality, assigns probability for selection
- Selection randomly chooses pairs for combinations depending on fitness
- Crossover point randomly chosen for each individual, offsprings are generated
- Mutation randomly changes a state