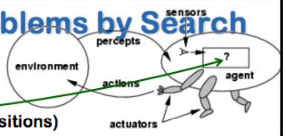
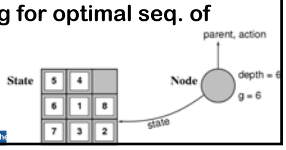
Artificial Intelligence

Prof. Björn Ommer
HCI & IWR

Review: Solving Problems by Search



- **Given:**
  - State space
  - possible actions (= state transitions)
- **Goal:**
  - reach a specific state (or set of states w/ certain charatceristics)
- **Solution:**
  - sequence of actions (leading from initial state to goal)
- **Find solution by searching for optimal seq. of actions**



Single-state problem formulation

A problem is defined by four items:

1. initial state e.g., "at Arad"
2. actions or successor function $S(x)$ = set of action–state pairs
   e.g., $S(Arad)$ = {<Arad → Zerind, Zerind>, … }
3. goal test, can be
   - explicit, e.g., x = "at Bucharest"
   - implicit, e.g., NoDirt(x)
4. path cost (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - $c(x,a,y)$ is the step cost, assumed to be ≥ 0

- A solution is a sequence of actions leading from the initial state to a goal state

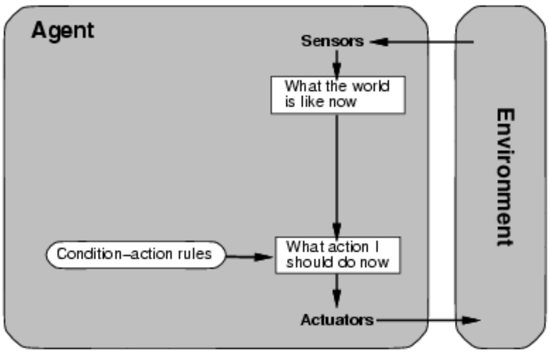
---

Review: Solving Problems by Search

- **Uninformed Search:**
  - only local info about present state
  - e.g., "in goal state?", "how much has agent earned up till now?"
- **Informed Search:**
  - Estimate of distance to goal from present state
- ➪ Desirability of
  - present state as is vs. based on potential successors of present state

---

Search-based Agent



Outline – Informed search algorithms

- Best-first search
- Greedy best-first search
- A* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms

Review: Tree search

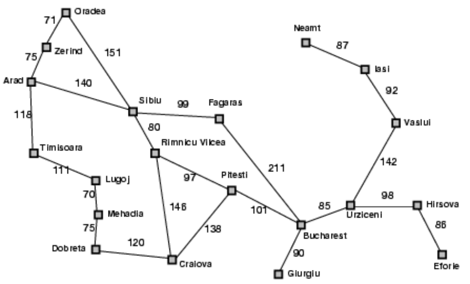
```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
      if fringe is empty then return failure
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST[problem] applied to STATE(node) succeeds return node
      fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

- A search strategy is defined by picking the order of node expansion

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - → Expand most desirable unexpanded node
- Implementation:
  Order the nodes in fringe in decreasing order of desirability
- Special cases:
  - greedy best-first search
  - A* search

Romania with step costs in km



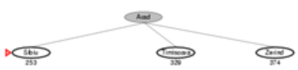
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

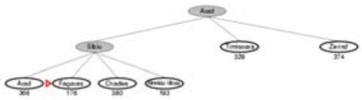
- Evaluation function *h(n)* (heuristic)
  = estimate of cost from *n* to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

Greedy best-first search example

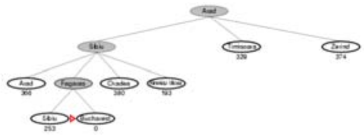
Greedy best-first search example



Greedy best-first search example

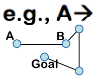


Greedy best-first search example



Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., A→ B → A → B → …



- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

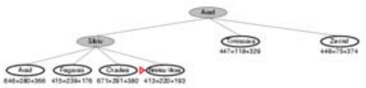
A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

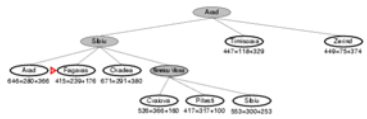
A* search example

A* search example

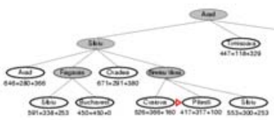
A* search example



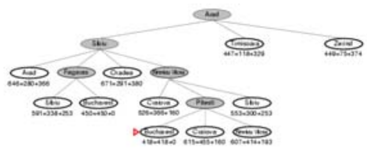
A* search example



A* search example



A* search example

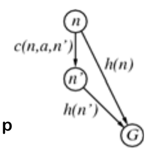


Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n,
  $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal

Consistent heuristics

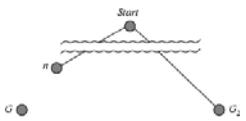
- A heuristic is consistent if for every node n, every successor n' of n generated by any action a,
  $h(n) \leq c(n,a,n') + h(n')$
- If h is consistent, we have
  $f(n') = g(n') + h(n')$
  $= g(n) + c(n,a,n') + h(n')$
  $\geq g(n) + h(n)$
  $= f(n)$



- i.e., $f(n)$ is non-decreasing along any p
- Theorem: If $h(n)$ is consistent, A* using GRAPH-SEARCH is optimal

Optimality of A* (proof)

- **Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node in the queue such that n is on a shortest path to an optimal goal G.**

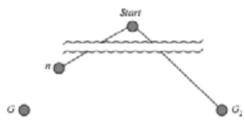


- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

UNIVERSITÄT HEIDELBERG

Optimality of A* (proof)

- **Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node in the queue such that n is on a shortest path to an optimal goal G.**



- $f(G_2) > f(G)$ from above
- $h(n) \le h^*(n)$ since h is admissible
- $g(n) + h(n) \le g(n) + h^*(n)$
- $f(n) \le f(G)$

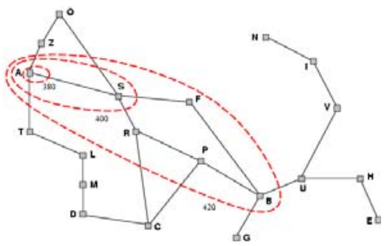
Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

UNIVERSITÄT HEIDELBERG

Optimality of A*

- **A* expands nodes in order of increasing *f* value**

- Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



UNIVERSITÄT HEIDELBERG

Properties of A*

- Complete? **Yes (unless there are infinitely many nodes with $f \le f(G)$)**
- Time? Exponential [relative error in $h *$ length of soln.]
- Space? Keeps all nodes in memory
- Optimal? Yes: cannot expand f_{i+1} until f_i is finished

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

…*Now:* improve speed by better heuristics

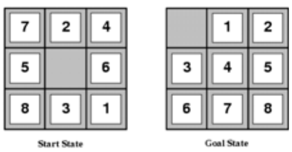
UNIVERSITÄT HEIDELBERG

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



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

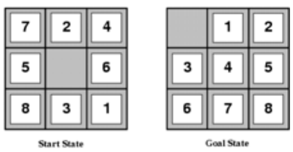
UNIVERSITÄT HEIDELBERG

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1 ⇨ h_2 is better for search
- Typical search costs (average number of nodes expanded):
  - *d=12* IDS = 3,644,035 nodes, $A^*(h_1)$ = 227 nodes, $A^*(h_2)$ = 73 nodes
  - *d=24* IDS = too many nodes, $A^*(h_1)$ = 39,135 nodes, $A^*(h_2)$ = 1,641 nodes
- Given any admissible heuristics h_a, h_b,
  $h(n) = \max(h_a(n); h_b(n))$
  is also admissible and dominates h_a, h_b

Relaxed Problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- Removing constraints (relaxation) creates extra edges to search graph

⇨ state space graph of relaxed problem is *super*graph of original one

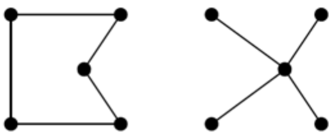
- Extra edges: optimal solution in original problem is also solution of relaxed problem
- But relaxed problem may have better solutions, since extra edges can provide shortcuts

Relaxed Problems

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
  - If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
- Key point: the optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem

Relaxed Problems contd.

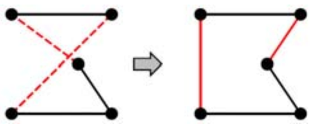
- Well-known example: travelling salesperson problem (TSP)
- Find the shortest tour visiting all cities exactly once



- Minimum spanning tree can be computed in $O(n^2)$ and is a lower bound on the shortest (open = exactly one visit) tour

Example: TSP

- Start with any complete tour, perform pairwise exchanges



- Variants of this approach get within 1% of optimal very quickly with thousands of cities

Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest h
  - incomplete and not always optimal
- A* search expands lowest g + h
  - complete and optimal
  - also optimally efficient (up to tie-breaks, for forward search)
- Admissible heuristics can be derived from exact solution of relaxed problems

Outline – Informed search algorithms

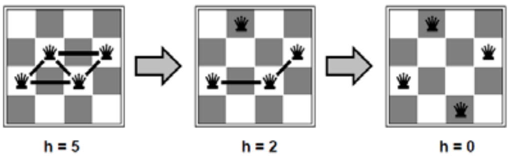
- Best-first search
- Greedy best-first search
- A* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

Example: *n*-queens

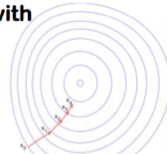
- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts



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

Hill-climbing search (gradient ascend)

- "Like climbing Everest in thick fog with amnesia"

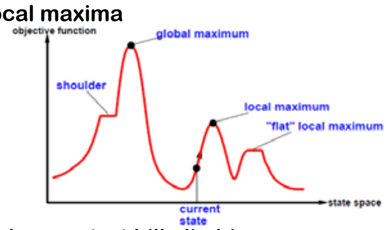


```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   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
```

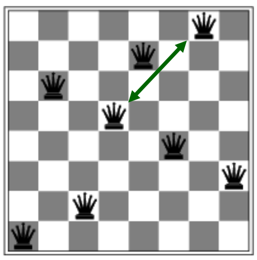

Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima



- Random-restart hill climbing overcomes local maxima; trivially complete
- Random sideways moves escape from shoulders loop on at maxima

Hill-climbing search: 8-queens problem



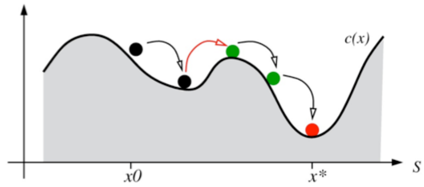
- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- A local minimum with h = 1

Simulated annealing search

- Most minimization strategies find the *nearest local minimum*
- Standard strategy:
  - Generate trial point based on current estimates
  - Evaluate function at proposed location
  - Accept new value if it improves solution
- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

Solution

- We need a strategy to find other minima
- This means, we must sometimes select new points that do not improve solution



- How?

Annealing

- One manner in which crystals are formed
- Gradual cooling of liquid …
  - At high temperatures, molecules move freely
  - At low temperatures, molecules are "stuck"
- If cooling is slow
  - Low energy, organized crystal lattice formed

Simulated Annealing

- Analogy with thermodynamics
- Incorporate a temperature parameter into the minimization procedure
- At high temperatures, explore parameter space
- At lower temperatures, restrict exploration
- Consider decreasing series of temperatures
- For each temperature, iterate these steps:
  - Propose an update and evaluate function
  - Accept updates that improve solution
  - Accept some updates that don't improve solution
  - Acceptance probability depends on "*temperature*" *parameter*

Example Application - TSP

- The traveling salesman problem
  - Salesman must visit every city in a set
  - Given distances between pairs of cities
  - Find the shortest route through the set
- No practical deterministic algorithms for finding optimal solution are known…
  - … simulated annealing and other stochastic methods can do quite well

Update Scheme

- A good scheme should be able to:
  - Connect any two possible paths
  - Propose improvements to good solutions
- Some possible update schemes:
  - Swap a pair of cities in current path
  - Invert a segment in current path

Example Trajectories in State Space

Fig. 9. Results at four temperatures for a clustered 400-city traveling salesman problem. The points are uniformly distributed in nine regions. (a) $T = 1.2$, $\alpha = 2.0567$; (b) $T = 0.8$, $\alpha = 1.515$; (c) $T = 0.4$, $\alpha = 1.055$; (d) $T = 0.0$, $\alpha = 0.7839$.

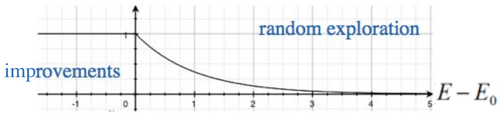
Updating

- **Metropolis (1953), Hastings (1970)**
  - Define a set of conditions that, if met, ensure the random walk will sample from probability distribution at equilibrium
    - In theory
- Recommendations apply to how changes are proposed and accepted

Accepting an Update

- The ***Metropolis criterion:*** Change from E_0 to E with probability

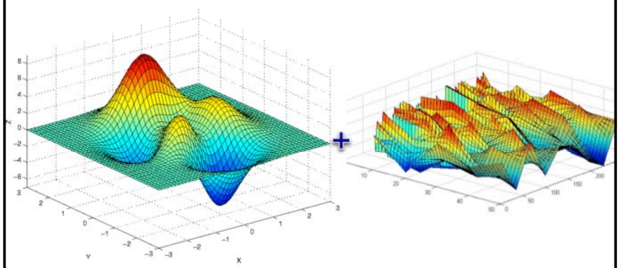
$$\min\left(1, \exp\left\{-\frac{(E-E_0)}{T}\right\}\right)$$



- Given sufficient time, leads to equilibrium state

Simulated Annealing and Gradient Descent

- **Perform gradient descent on original cost fct +** random noise (noise energy prop. to temp T)



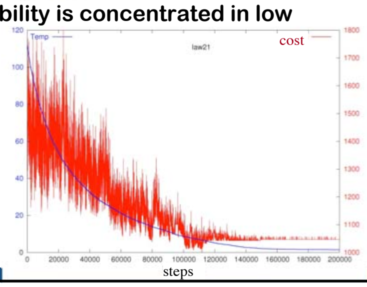
Key Requirement: *Irreducibility*

- All states must communicate
  - We can get everywhere from any arbitrary starting point
  - Starting point should not affect results
- If Q is matrix of proposal probabilities
  - Either Q_ij > 0 for all possible states i and j
  - Some integer P exists where (Q^P)_ij > 0 for all *i,j*
    - *After P steps we can reach any j from any i*

- Probability of state with energy *k is*

$$P(E=k) \propto \exp\left(-\frac{k}{T}\right)$$

- At low T, probability is concentrated in low energy states



Simulated Annealing Recipe

1. Select starting temperature and initial parameter values
2. Randomly select a new point in the neighborhood of the original
3. Compare the two points using the *Metropolis criterion*
4. Repeat steps 2 and 3 until system reaches equilibrium state…
   - In practice, repeat the process N times for large N
5. Decrease temperature and repeat the above steps, stop when system reaches frozen state

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
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, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(Δ E/T)
```


Practical Issues

- The maximum temperature
- Scheme for decreasing temperature
  - Cooling schedules
- Strategy for proposing updates
  - E.g.: Select a component to update

    & Sample from within plausible range

Properties of simulated annealing search

- At fixed "temperature" T, state occupation probability reaches Boltzmann distribution

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

- T decreased slowly enough ⇨ 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

- However, annealing can be exponentially slow
- Devised by Metropolis et al., 1953, for physical process modelling
- Widely used in VLSI layout, airline scheduling, etc

Local beam search

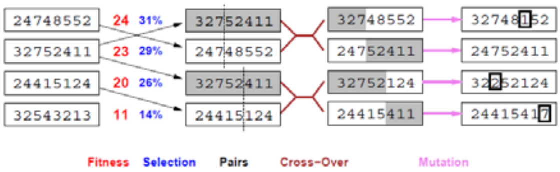
- Keep track of *k* states rather than just 1; choose top k of all their successors
  - Start with *k* randomly generated states
  - At each iteration, all the successors of all *k* states are generated
  - Not the same as k searches run in parallel! Searches that find good states recruit other searches to join them
  - If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.
- Observe the close analogy to natural selection!

Genetic algorithms

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation
- If there is more info on cost fct, more effective optimization strategies are typically preferred

Genetic algorithms

= stochastic local beam search + generate successors from pairs of states

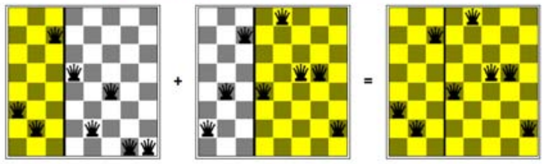


- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms

GAs require states encoded as strings (GPs use programs)

Crossover helps iff substrings are meaningful components



GAs $\neq$ evolution: e.g., real genes encode replication machinery!