

Informed search

•Russell and Norvig, Chapters 3 and 4

Next Week: Wednesday Tutorial

- Ekka holiday 17 August
- Next week, people signed on to the Wednesday tutorial should either attend another tutorial or submit online by 5pm on Thursday

Psychobabble: A Podcast of Experimental Psychology

- www.psycho-babble.net
- Ep14 – AI and the Spatial Universe



www.ai-class.com



Overview: aims

- understand what best-first search involves
- know the Greedy search technique and its properties
- know the A* search technique and its properties
- be able to apply and design heuristic functions, understand admissibility in this context
- be familiar with local search/iterative improvement algorithms, e.g. hill-climbing and simulated annealing

Overview: topics

- Using heuristics
- Best-first search
- Iterative improvement search

Breadth-first search complexity...

- With $b=10$
- Assume that 1 node takes 1ms to expand and 100 bytes to store (main memory)

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Uninformed Search

- Find solutions by generating new states and testing them against the goal state
- Not domain specific
- States are either start, goal, or other
- Inefficient in most cases
- Alternatives
 - Informed (heuristic) search algorithms
 - Local search algorithms

Non-exhaustive search

- Delay exploring “non-promising” parts of the tree
 - reasonable answer in reasonable time
- Domain specific
- Informed search strategies
 - node n is selected for expansion based on an evaluation function $f(n)$
 - knowledge of domain encoded in heuristic function
 - **$h(n)$ – estimated cost from n to nearest goal**

Heuristic Functions

- Estimate of goodness
- Use to inform the search
- Should be admissible, i.e. never overestimate the cost to goal
- Some heuristics are better than others
 - Those that are not very good will require more search
- Any easy to calculate heuristic is better than none
- Heuristics cannot be perfect
- The ‘best’ value for a heuristic is the cost to reach the goal, but in order to find this an exhaustive search is required

Heuristic Search

- *The time spent evaluating the heuristic function in order to select a node for expansion must be recovered by a corresponding reduction in the size of the search space explored.*
- Base-level / meta-level trade-off
- (solving the problem / deciding what to do)

Heuristic function

- Evaluation function
 - $f(n)$ = some combination of $g(n)$ and $h(n)$
 - $g(n)$ = cost so far to reach n : known
 - $h(n)$ = estimated cost to goal from n : heuristic function
 - $f(n)$ = estimated total cost of path through n to goal

Best-first search

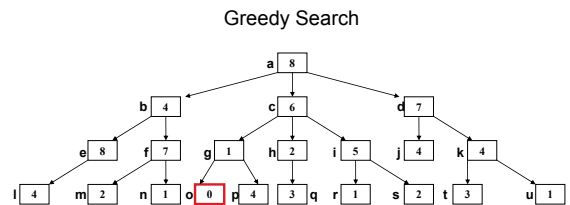
- Expand that node in list which is evaluated as “best”
- Effectiveness depends on quality of guesses – different heuristic functions
- Implemented with a priority queue
 - Maintain the fringe in ascending order of f -values
- Two kinds of best-first algorithms:
 - Greedy search: Expand node closest to goal; choose node n where $f(n) = h(n)$ is minimised
 - A*: Expand node on least-cost solution path; choose n where $f(n) = h(n) + g(n)$ is minimised
 - Maths notation: choose $n^1 = \arg \min_n f(n)$, n in fringe

Greedy search

- Tries to expand the node closest to the goal
- Idea: use an evaluation function for each node
 - Estimate of desirability
 - \Rightarrow expand most desirable node
- Implementation
 - Fringe is a queue sorted in decreasing order of desirability

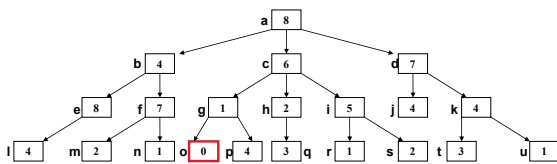
Greedy search

1. Set L to be a list of the initial nodes in the problem.
2. Let n be the first node on L that is expected to be closest to the goal. If L is empty, fail.
3. If n is a goal node, stop and return it and the path from the initial node to n .
4. Otherwise, remove n from L and add to L all of n 's children, labelling each with its path from the initial node. Return to step 2.



Using *greedy search* and given these heuristic values determine the order in which the nodes are searched.

Greedy Search



List

$h(a)=8$

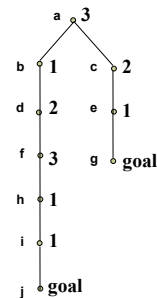
$h(b)=4$ $h(c)=6$ $h(d)=7$

$h(c)=6$ $h(d)=7$ $h(f)=7$ $h(e)=8$

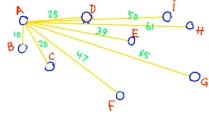
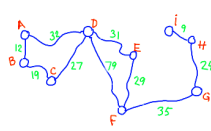
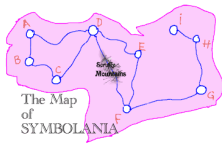
$h(g)=1$ $h(h)=2$ $h(i)=5$ $h(d)=7$ $h(f)=7$ $h(e)=8$

$h(o)=0$ $h(h)=2$ $h(p)=4$ $h(i)=5$ $h(d)=7$ $h(f)=7$ $h(e)=8$

Using a greedy search procedure, work out the order in which the nodes are expanded in this search space.

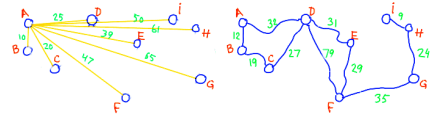


Out-and-about in Symbolonia (1)



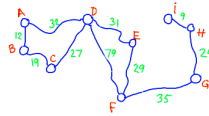
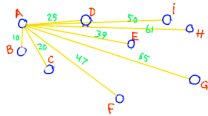
- The country of Symbolonia has 9 major cities, neatly named after the first 9 letters in the alphabet (red).
- There is a network of roads connecting the cities. The distances are provided in green. Note that the Scuffy Mountains make the road from F to D much longer than expected.
- We need to find our way by car from our present location to the City of A (A has an airport that will take us home). From a birds-eye view the distances are as in the illustration (again green numbers). This is the kind of information that we have available with respect to the goal at all states (assuming that we know the coordinates of A and the city that we're in).
- So our heuristic function, $h()$, returns the Euclidean distance as shown above, e.g. $h(F)=47$, $h(H)=61$, etc.

Greedy Search in Symbolonia (1)



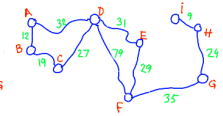
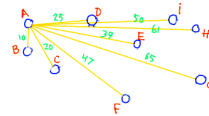
- E to A
- $h(E)=39$
- $h(D)=25$ $h(F)=47$
- $h(A)=0$ $h(C)=20$ $h(E)=39$ $h(F)=47$
- -> Path = EDA

Greedy Search in Symbolonia (2)



- F to A
- $h(F)=47$
- $h(D)=25$ $h(E)=39$ $h(G)=65$
- $h(A)=0$ $h(C)=20$ $h(E)=39$ $h(F)=47$ $h(G)=65$
- -> Path = FDA (optimal path is FEDA)
Greedy search is not *optimal*

Greedy Search in Symbolonia (3)



- H to A
- $h(H)=61$
- $h(I)=50$ $h(G)=65$
- $h(H)=61$ $h(G)=65$
- $h(I)=50$ $h(G)=65$
- ... Greedy search is not *complete*

Properties of Greedy search

- Complete
 - No - can be stuck in loops
 - Complete in finite space with repeated-state checking
- Time
 - $O(b^m)$ but a good heuristic can give dramatic improvement
- Space
 - $O(b^m)$ keeps all nodes in memory
- Optimal
 - No

Greedy search is not optimal, but is often efficient

Refined search objective

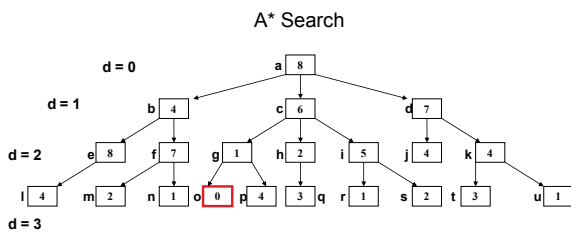
- To find the shallowest goal (i.e. the best goal) as quickly as possible
- Expand the node that appears to be closest to a shallow goal
- A* algorithm

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 to goal from n
 - $f(n)$ = estimated total cost of path through n to goal

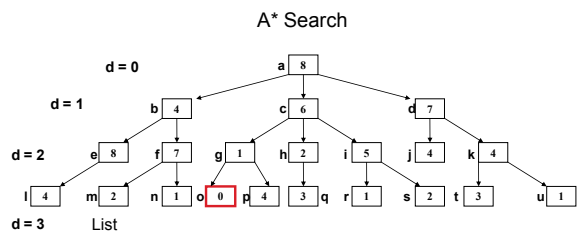
A* Algorithm

- Set L to be a list of the initial nodes in the problem.
- Let n be the node on L for which $f(n)$ is minimal. If L is empty, fail.
- If n is a goal node, stop and return it and the path from the initial node to n .
- Otherwise, remove n from L and add to L all of n 's children, labelling each with its path from the initial node. Return to step 2.



Using A* search and given these heuristic values determine the order in which the nodes are searched.

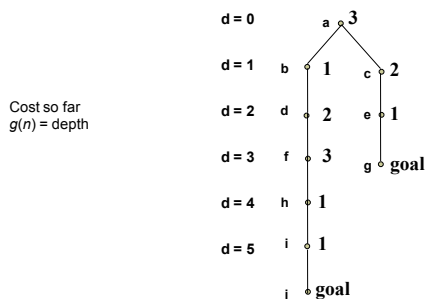
Cost so far
 $g(n)$ = depth



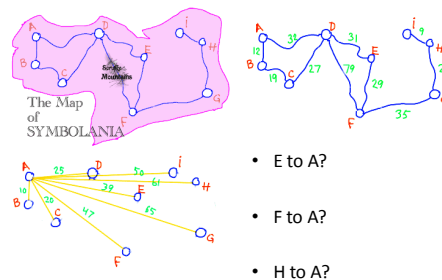
List

$f(a)=0+8$
 $f(b)=1+4=5$ $f(c)=1+6=7$ $f(d)=1+7=8$
 $f(e)=7$ $f(f)=8$ $f(g)=2+7=9$ $f(h)=2+8=10$
 $f(i)=2+1=3$ $f(j)=2+2=4$ $f(k)=2+5=7$ $f(l)=8$ $f(m)=9$ $f(n)=10$
 $f(o)=3+0=3$ $f(p)=4$ $f(q)=3+4=7$ $f(r)=7$ $f(s)=8$ $f(t)=9$ $f(u)=10$

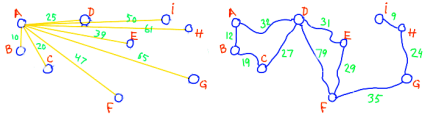
Given the original estimations of $h(n)$, use the A* algorithm, work out the new cost of each node and the subsequent order in which the nodes are expanded.



Out-and-about in Symbolania (2)

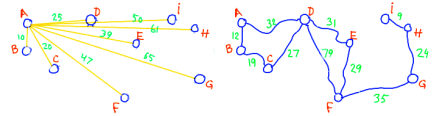


A* Search in Symbolania (1)



- E to A
- $f(E)=(0+39)=39$
- $f(D)=(31+25)=56$ $f(F)=(29+47)=76$
- $f(A)=(31+32+0)=63$ $f(C)=(31+27+20)=78$
- $f(E)=(31+31+39)=101$ $f(F)=76$
- $f(F)=(31+79+47)=157$
- -> Path = EDA

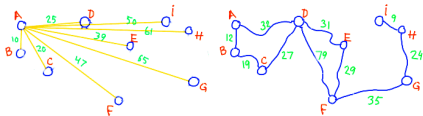
A* Search in Symbolania (2)



- F to A
- $f(F)=(0+47)=47$
- $f(E)=(29+39)=68$ $f(G)=(35+65)=100$ $f(D)=(79+25)=104$
- $f(D)=(29+31+25)=85$ $f(G)=(35+65)=100$ $f(D)=(79+25)=104$
- $f(F)=(29+29+47)=105$
- $f(A)=(29+31+32+0)=92$ $f(G)=(35+65)=100$ $f(D)=(79+25)=104$
- $h(C)=(29+31+27+20)=107$ $h(E)=(29+31+31+39)=130$
- $h(F)=(29+31+79+47)=186$
- -> Path = FEDA

A* search is *optimal*

A* Search in Symbolania (3)



- H to A
- $f(H)=(0+61)=61$
- $f(I)=(9+50)=59$ $f(G)=(24+65)=89$
- $f(H)=(9+9+61)=79$ $f(G)=89$
- $f(I)=(9+9+9+50)=77$ $f(G)=89$ $f(G)=(9+9+24+65)=107$
- $f(G)=89$ $f(H)=(9+9+9+9+61)=97$ $f(G)=107$
- $f(H)=97$ $f(F)=(24+35+47)=106$ $f(G)=107$ $f(H)=(24+24+61)=109$
- ...

Might take a while but A* search is *complete*

Properties of A*

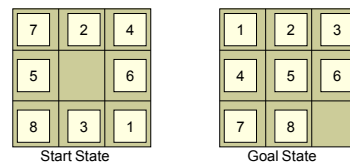
- If $h(n)$ is an admissible heuristic function, A* search is complete and optimal
- However for most problems, the number of nodes within the search space is exponential in the length of the solution
- Computational time is not the main drawback
 - Keeps all generated nodes in memory
 - A* usually runs out of space long before it runs out of time!

Best-first summary

- Best-first search expands minimum cost nodes (according to some measure) first
- **Greedy search** minimises the estimated cost to the goal, $f(n)$
 - Usually decreases search time
 - Neither complete nor optimal
- **A* search** is complete & optimal, but can have prohibitive space/time complexity

Heuristic functions

- Example: Heuristics for the 8-puzzle



- Average solution cost for a randomly generated 8-puzzle is about 22 steps
 - Example above: solution is 26 steps
- Branching factor is approximately 3

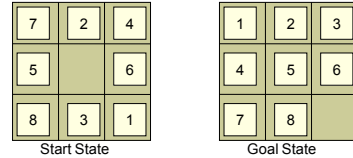
Admissible Heuristics (1)

- $h(n)$ must never overestimate the cost to reach the goal
- Admissible heuristics are optimistic
 - They think the cost of solving the problem is less than it actually is
- A* is optimal if $h(n)$ is an admissible heuristic
- Example admissible heuristic
 - Straight line distance used to get to Brisbane



Admissible Heuristics (2)

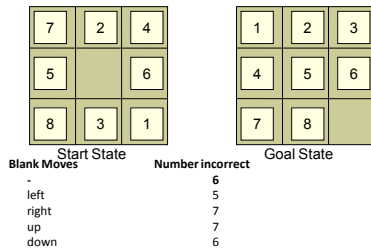
- If we want to find the shortest solution using A* we need a heuristic function that never overestimates the number of steps to the goal



- Average solution cost: 22 steps

An 8-puzzle heuristic method

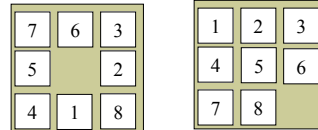
- Heuristic method
 - Count the number of tiles in the wrong place
- No guarantees of success but useful



8-puzzle heuristic function h_1

- For the 8-puzzle
 - How do we translate our heuristic method – count the number of tiles in the wrong place – into a function that our computer program can use?

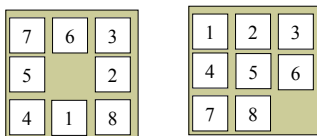
- $h_1(n)$ = number of misplaced tiles



- $h_1(S) = 7$
- Admissible – any tile out of place would have to be moved at least once

8-puzzle heuristic function h_2

- $h_2(n)$ = total Manhattan/city-block distance
 - i.e. number of squares from the desired location of each tile



- $h_2(S) = 3+2+0+1+1+2+2+1 = 12$
- Admissible – all any move can do is move one tile closer to the goal

Heuristic Performance

- Quality of a heuristic may be measured by the **effective branching factor**
- If the number of nodes generated by A* for a particular problem is N , and the solution is at depth d , then b^* is the branching factor that a uniform tree of depth d would have to have in order to contain $N+1$ nodes

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

- Well designed heuristic would have a value of b^* close to 1

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then $h_2(n)$ **dominates** $h_1(n)$ and is better for search
- It is always better to use an admissible heuristic function with higher values
- Below is the cost and b^* for 8-puzzle solutions

d	Search Cost			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3096	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

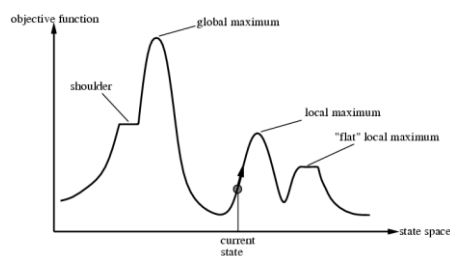
Inventing admissible heuristics

- For 8-puzzle h_1 and h_2 are accurate path lengths for simplified version of puzzle
- Problem with fewer restrictions on actions is a relaxed problem
- 8-puzzle actions:
 - A tile can move from square A to square B if A is adjacent to B and B is blank
 - Three relaxed versions of the above action-rule
 1. a tile can move from A to B if A is adjacent to B
 2. a tile can move from A to B if B is blank
 3. a tile can move from A to B
 - From 1. we can derive h_2
 - From 3. we can derive h_1

Local search algorithms: Iterative Improvement

- Often state description contains all of the information needed
 - Path is irrelevant – we no longer care about the steps to get there
 - Only “local” information is used – the space is not searched with a “global” view
- Iterative improvement: start in a complete configuration and make incremental changes to improve its quality
- Improvement can be measured using an objective function

Aim: find the global maximum by modifying the current state from local information



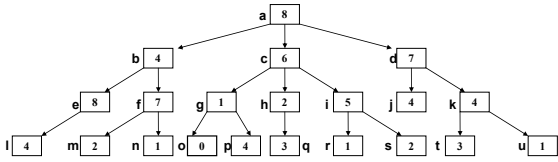
Three local search algorithms

- Hill climbing
 - Take actions that improve the current state (greedy local search)
- Simulated annealing
 - Select actions stochastically (may make things worse temporarily) according to a decreasing “temperature”
- Genetic algorithms
 - Search using a population of states, in which each state is subjected to “evolutionary selection” and “evolutionary processes” to create a new (more “fit”) population of states

Hill Climbing search

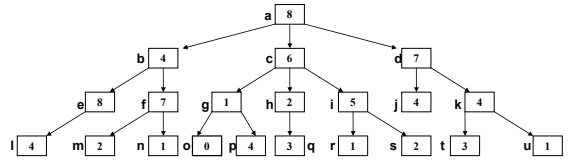
1. Set n be the initial node in the problem.
2. If n is a goal node, stop and return it.
3. Otherwise, expand n (consider only the successors of n). Use heuristic function to sort n 's children by their expected distance to the goal (greedy search).
4. Assign n the value of the best successor.
Return to step 2.

Hill Climbing



Using hill-climbing search and given these heuristic values determine the order in which the nodes are searched.

Hill Climbing

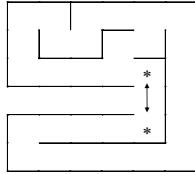


$F(a)=8$
 $F(b)=4$ $F(c)=6$ $F(d)=7$
 $F(f)=7$ $F(e)=8$. Stop: best solution: b, with $F(b)=4$.

Improvements: random re-starts, random move order, sideways moves.

Problems

- Problems of local maxima (if maximising)
- Dealing with plateaux
- Ridges



Variations of Hill Climbing

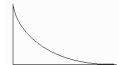
- Stochastic hill climbing
 - Randomly choose one of the uphill moves
- First-choice hill climbing
 - Randomly generative successor states, choose first that is better than the current state
- Random-restart hill climbing
 - Start from a series of randomly generated initial states

Simulated Annealing (1)

- Named after a metal-casting technique
 - molten metal is gradually *annealed*
 - gradual temperature decrease results in a low energy structure
 - analogous to a lost cost solution
- Purpose - to avoid the problem of local minima
- Propose steps in random directions
- Accept moves which lower cost and some which increase it

Simulated Annealing (2)

- Temperature parameter controls how often we accept moves which increase cost
- Usually given a schedule like: $T(x) = \exp(-x)$, where x = number of moves proposed



- Initially: accept almost any move
- Finally: only accept improvements (same as hill-climbing)
- Moves smoothly from global to local search
- Avoids local minima at various scales

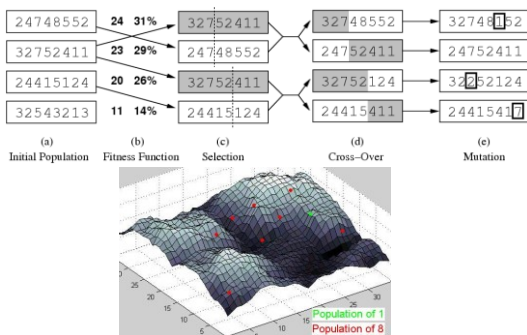
Simulated Annealing (3)

- Idea: escape local minima by allowing some “bad” moves but *gradually decrease their probability*
- A probabilistic element included
 - allows movement to worse states
- Greater probability of accepting worsening moves at the start
- Probability decreases slowly over time
- “Temperature” parameter: goes from high to low

Simulated Annealing (4)

1. Set n be the initial node in the problem.
2. If n is a goal node, stop and return it.
3. Otherwise, expand n (consider only the successors of n).
4. Assign m the value of one of n 's children, chosen randomly.
5. If m is better than n , assign n the value m . Else, assign n the value m with probability proportional to the temperature.
6. Decrease the temperature. Return to step 2.

The Genetic algorithm



Example: Using the genetic algorithm for “circuit design”

- Objective function is the inverse of the “energy consumed” by the design
1. A population of random designs are generated (we use numbers to specify the design)
 2. The population is “ranked” according to the objective function
 3. The best are stochastically selected as “parents”
 4. “Parents” (their design specifications) are stochastically “recombined” and “mutated” to generate a new population.
 5. Repeat from step 2 until one individual in the population is “good enough”

Summary

- Best-first search: Greedy search and A* search
- Heuristics – inventing admissible heuristics
- Problems with these methods
- Local search: Hill climbing, Simulated annealing, Genetic algorithms
- Applications