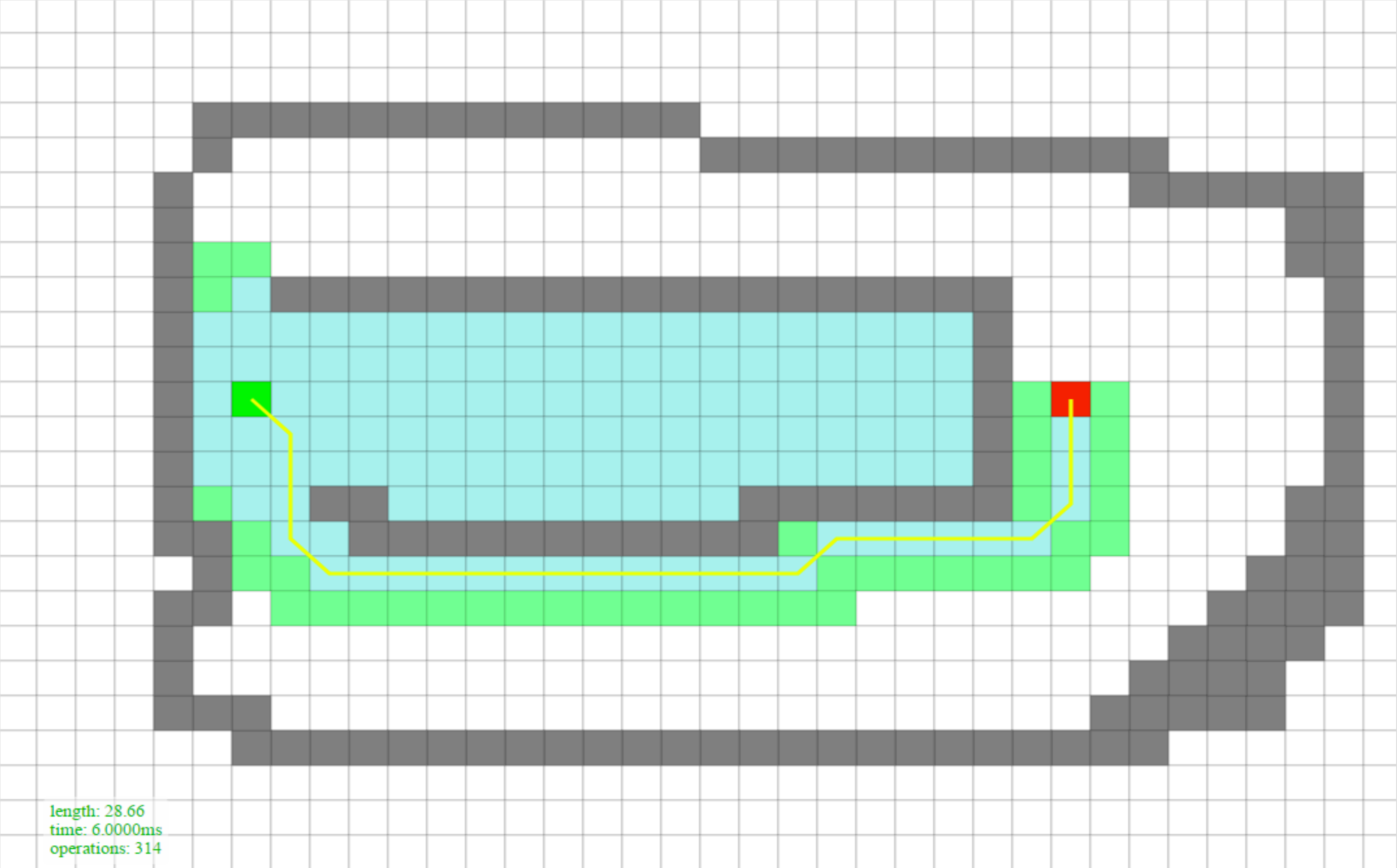
Informed Search (Ch. 3.5-3.6)



Announcements

Please raise hands when asking questions
(I will try to be better at looking for them)

Written assignment 1 is posted
- read a research paper
- use latex
(quick tutorial, see assignment pdf for links)

Heuristics

However, for A* to be optimal the heuristic h(node) needs to be...

For trees: <u>admissible</u> which means:

- h(node) $\leq$ optimal path from h to goal
- (i.e. h(node) is an underestimate of cost)

For graphs: <u>consistent</u> which means:

- h(node) $\leq$ cost(node to child) + h(child)
- (i.e. triangle inequality holds true)
- (i.e. along any path, f-cost increases)

Heuristics

Consistent heuristics are always admissible
-Requirement: h(goal) = 0

Admissible heuristics **might not** be consistent

A* is guaranteed to find optimal solution
if the heuristic is admissible for trees
(consistent for graphs)

Heuristics

In our example, the h(node) was the straight line distance from node to goal

This is an underestimate as physical roads cannot be shorter than this
(it also satisfies the triangle inequality)

Thus this heuristic is admissible
(and consistent)

Heuristics

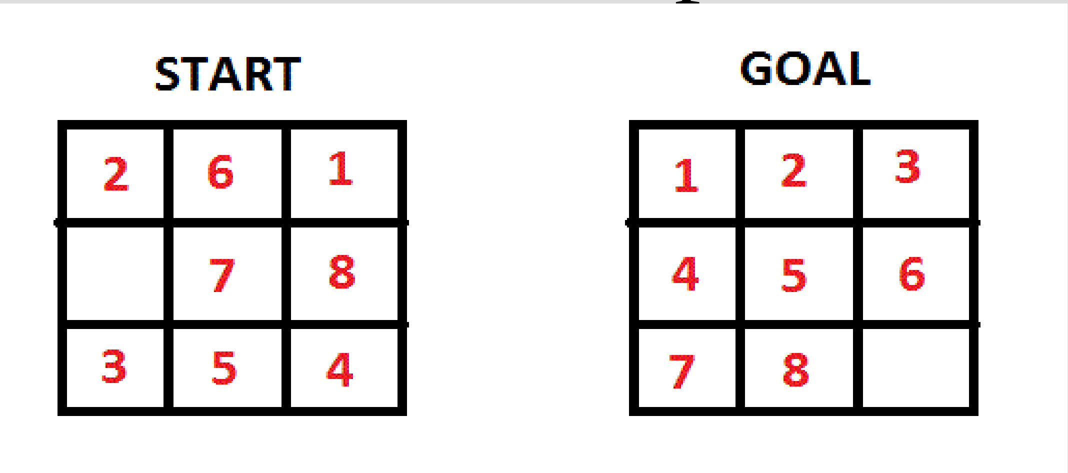
The straight line cost works for distances in the physical world, do any others exist?

One way to make heuristics is to <u>relax</u> the problem (i.e. simplify in a useful way)

The optimal path cost in the relaxed problem can be a heuristic for the original problem (i.e. if we were not constrained to driving on roads, we could take the straight line path)

Heuristics

Let us look at 8-puzzle heuristics:



The rules of the game are:

- You can swap any square with the blank

Relaxed rules:

1. Teleport any square to any destination
2. Move any square 1 space (overlapping ok)

Heuristics

1. Teleport any square to any destination
Optimal path cost is the number of mismatched squares (blank included)

2. Move any square 1 space (overlapping ok)
Optimal path cost is Manhattan distance for each square to goal summed up

Which ones is better? (Note: these optimal solutions in relaxed need to be computed fast)

Heuristics

h1 = mismatch count
h2 = number to goal difference sum

	Search Cost			Effective Branching Factor		
d	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	–	3056	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

Heuristics

The real branching factor in the 8-puzzle:

- 2 if in a corner
- 3 if on a side
- 4 if in the center
- (Thus larger “8-puzzles” tend to 4)

An <u>effective branching factor</u> finds the “average” branching factor of a tree (smaller branching = less searching)

Heuristics

The effective branching factor is defined as:

$$N = b^* + (b^*)^2 + (b^*)^3 + ... + (b^*)^d$$

... where:

N = the number of nodes (i.e. size of fringe + size of explored if tree search)

b^* = effective branching factor (to find)

d = depth of solution

No easy formula, but can approximate:

$$N^{1/(d+1)} \leq b^* \leq N^{1/d}$$

Heuristics

h2 has a better branching factor than h1, and this is not a coincidence...

h2(node) $\geq$ h1(node) for all nodes, thus we say h2 <u>dominates</u> h1 (and will thus perform better)

If there are multiple non-dominating heuristics: h1, h2... Then h* = max(h1, h2, ...) will dominate h1, h2, ... and will also be admissible /consistent if h1, h2 ... are as well

Heuristics

If larger is better, why do we not just set $h(node) = 9001$?

Heuristics

If larger is better, why do we not just set h(node) = 9001?

This would (probably) not be admissible...

If h(node) = 0, then you are doing the uninformed uniform cost search

If h(node) = optimal_cost(node to goal) then will ONLY explore nodes on an optimal path

Heuristics

You cannot add two heuristics ($h^* = h1 + h2$), unless there is no overlap (i.e. h1 cost is independent of h2 cost)

For example, in the 8-puzzles:

h3: number of 1, 2, 3, 4 that are misplaced

h4: number of 5, 6, 7, 8 that are misplaced

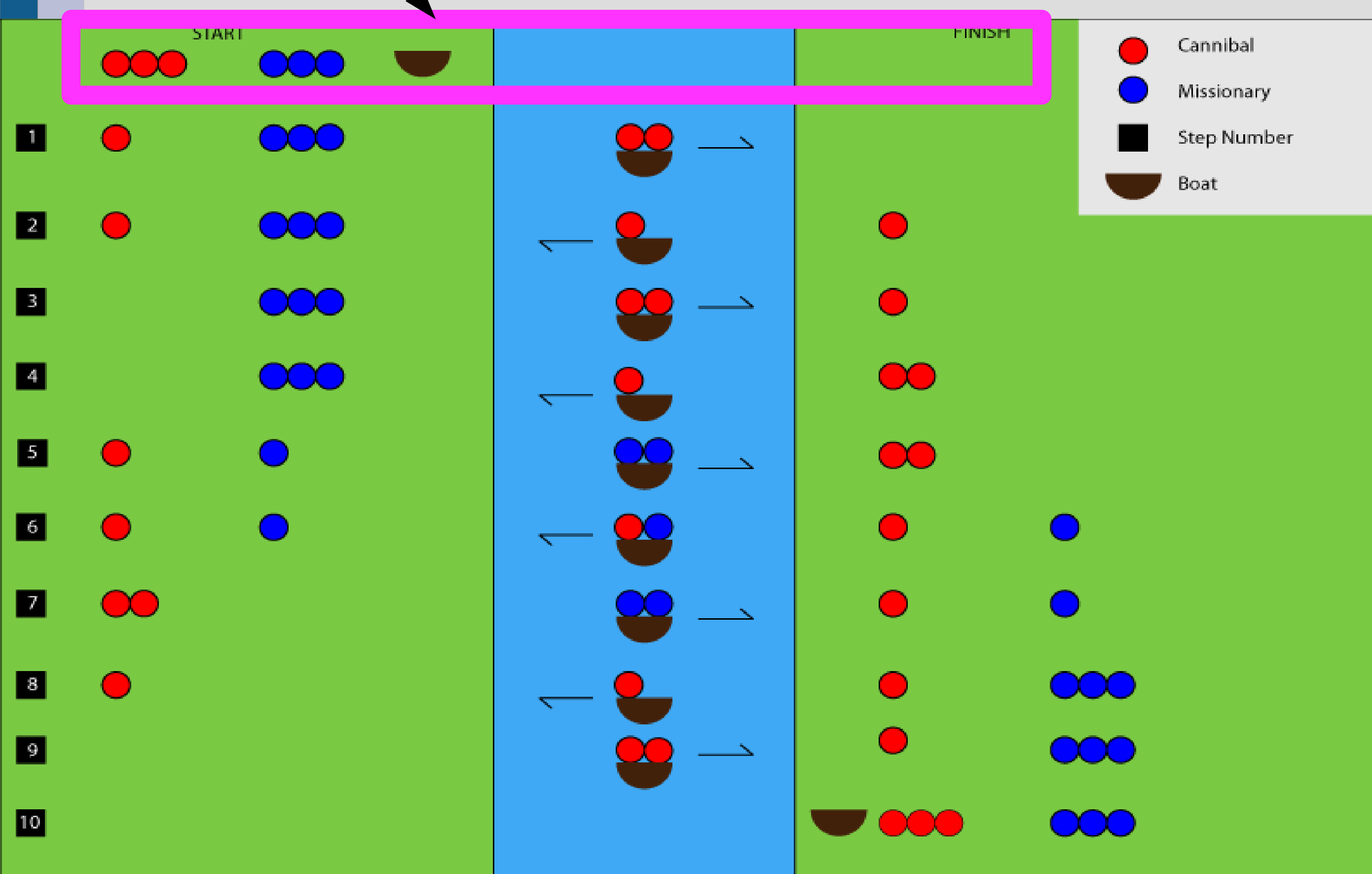
There is no overlap, and in fact:

$h3 + h4 = h1$ (as defined earlier)

Heuristics

Cannibals & missionaries problem:

initial



Goal:fewest steps to swap banks

Rules:

1. Either bank: m=<c, if m>0

2. 2 ppl in boat

3. Start: 3m & 3c

4. Need 1 in boat to move

Heuristics

What relaxation did you use? (sample)

Make a heuristic for this problem

Is the heuristic admissible/consistent?

Heuristics

What relaxation did you use? (sample)

- Remove needing person in boat to move

Make a heuristic for this problem

- h1 = [num people wrong bank]/2 (boat cap.)

Is the heuristic admissible/consistent?

- YES! The point of relaxing guarantees admissibility!

Local Search (Ch. 4-4.1)

Local search

Before we tried to find a path from the start state to a goal state

Now we will look at algorithms that do not care about the path, just try to find the goal

Some problems, may not have a clear “best” goal, yet we have some way of evaluating the state (how “good” is a state)

Local search

Today we will talk about 4 (more) algorithms:

1. Hill climbing
2. Simulated annealing
3. Beam search
4. Genetic algorithms

All of these will only consider neighbors while looking for a goal

Local search

These algorithms will also only consider the actions from their current state (neighbors)

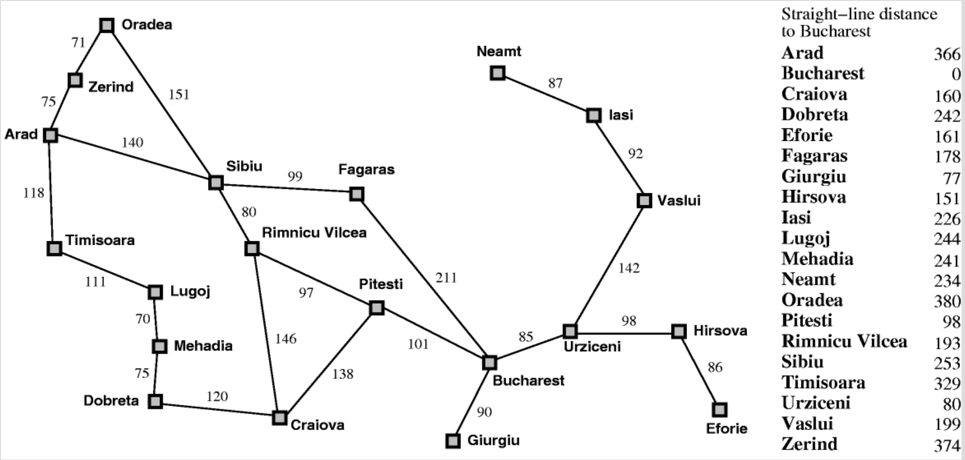
They all have a greedy component, along with typically a random component

In general, they can efficiently find a good solution, but have difficulty finding the best

Hill climbing

Remember greedy best-first search?

1. Pick child with best heuristic
2. Repeat 1...



Hill climbing is only a slight variation:

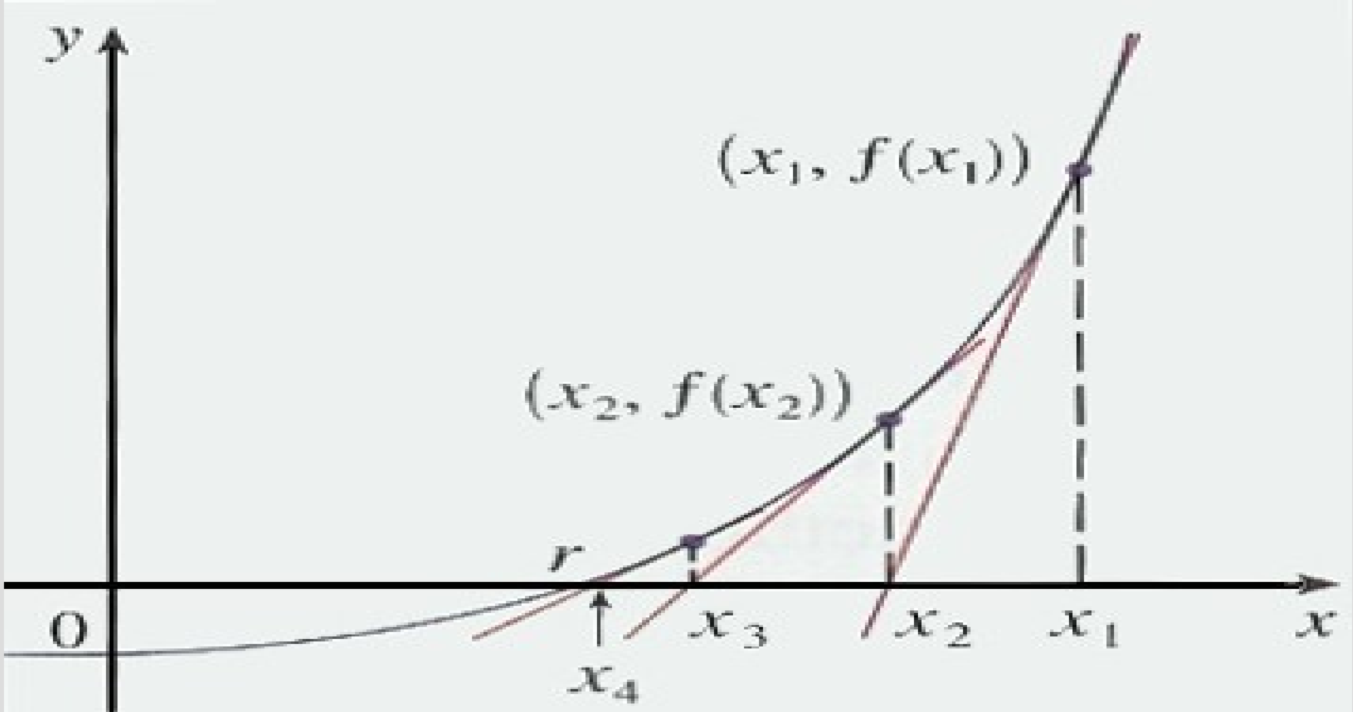
1. Pick best between: yourself and child
2. Repeat 1...

This avoids the looping issue...

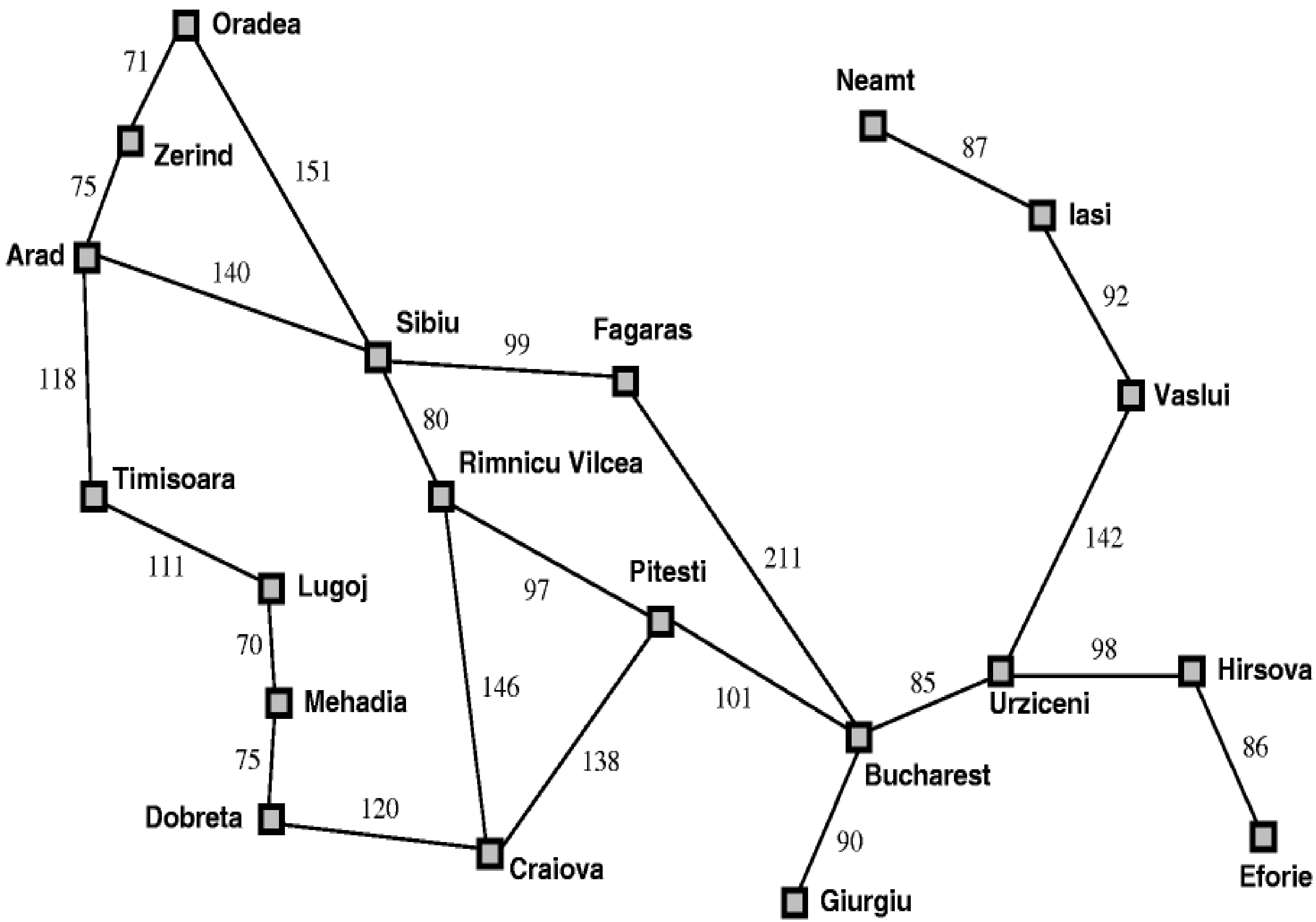
Hill climbing

This actually works surprisingly well, if getting "close" to the goal is sufficient (and actions are not too restrictive)

Newton's method:



Hill climbing



Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

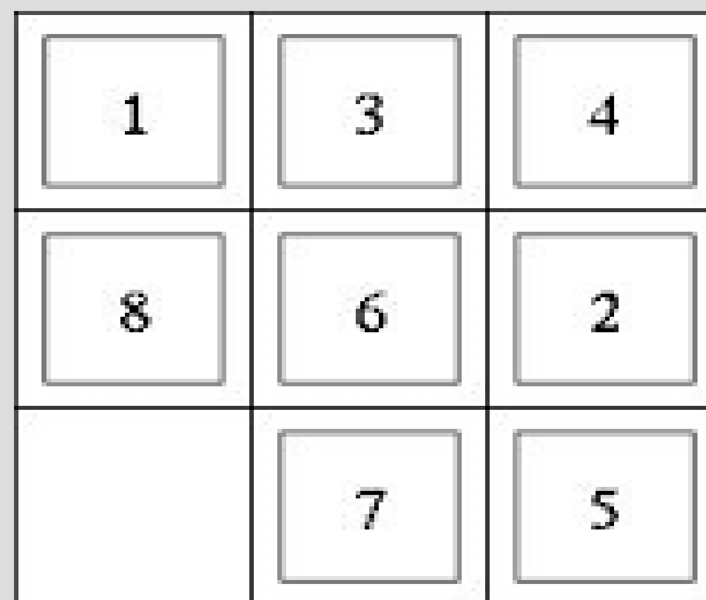
Hill climbing

For the 8-puzzles we had 2 (consistent) heuristics:

h1 - number of mismatched pieces

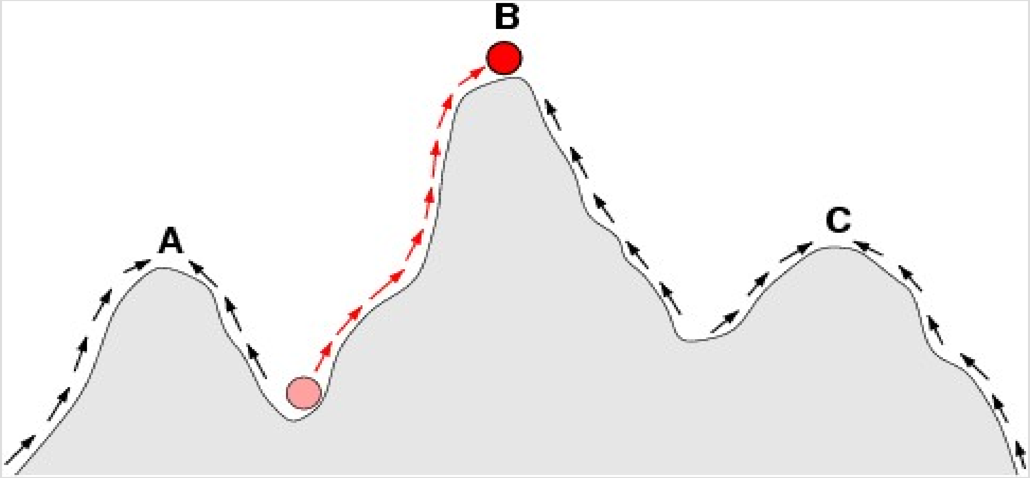
h2 - ∑ Manhattan distance from number's current to goal position

Let's try hill climbing this problem!

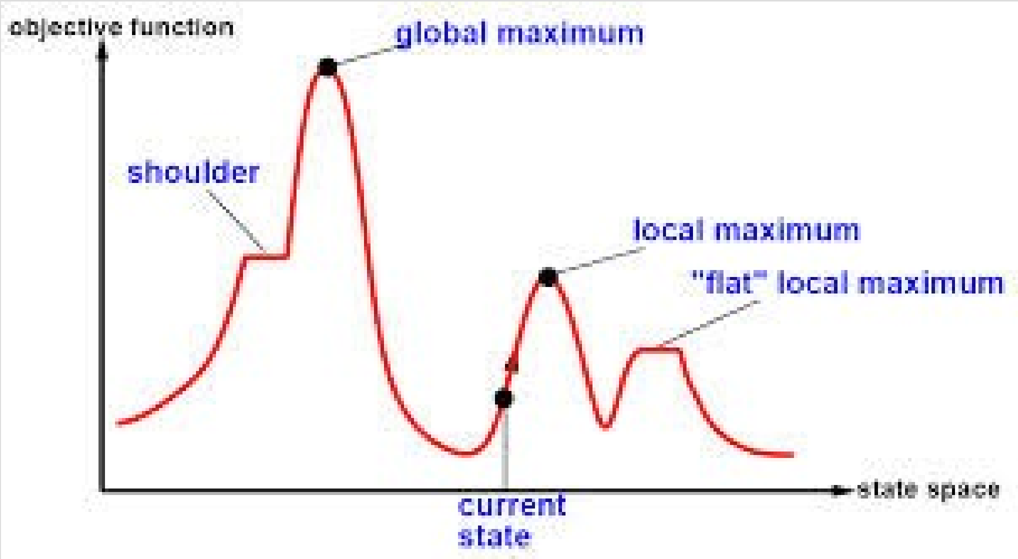


Hill climbing

Can get stuck in:
- Local maximum
- Plateu/shoulder



Local maximum will have a range of attraction around it



Can get an infinite loop in a plateu if not careful (step count)

Hill climbing

To avoid these pitfalls, most local searches incorporate some form of randomness

Hill search variants:

Stochastic hill climbing - choose random move and take that if better than current

Random-restart hill search - run hill search until maximum found (or looping), then start at another random spot and repeat

Simulated annealing

The idea behind simulated annealing is we act more random at the start (to “explore”), then take greedy choices later

https://www.youtube.com/watch?v=qfD3cmQbn28

An analogy might be a hard boiled egg:

1. To crack the shell you hit rather hard (not too hard!)
2. You then hit lightly to create a cracked area around first
3. Carefully peal the rest

Simulated annealing

The process is:

1. Pick random action and evaluation result
2. If result better than current, take it
3. If result worse accept probabilistically
4. Decrease acceptance chance in step 3
5. Repeat...

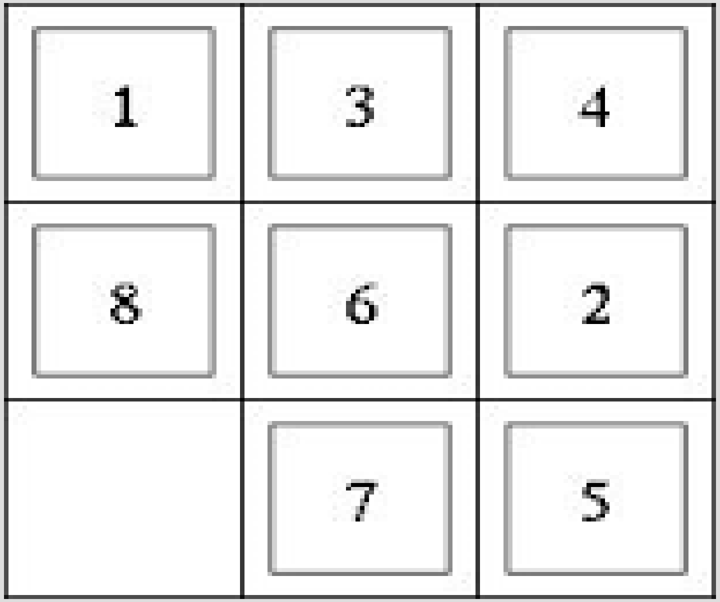
(see: SAacceptance.cpp)

Specifically, we track some “temperature” T:

3. Accept with probability: $e^{\frac{result-current}{T}}$
4. Decrease T (linear? hard to find best...)

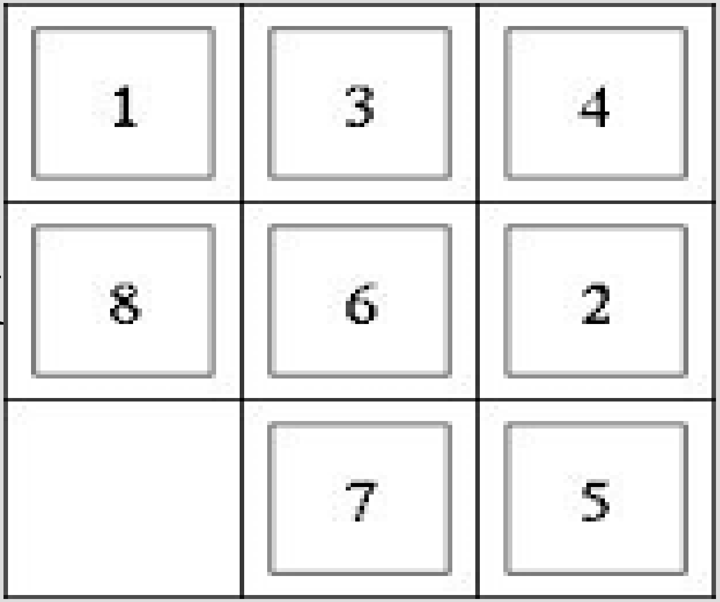
Simulated annealing

Let's try SA on 8-puzzle:



Simulated annealing

Let's try SA on 8-puzzle:



This example did not work well, but probably due to the temperature handling

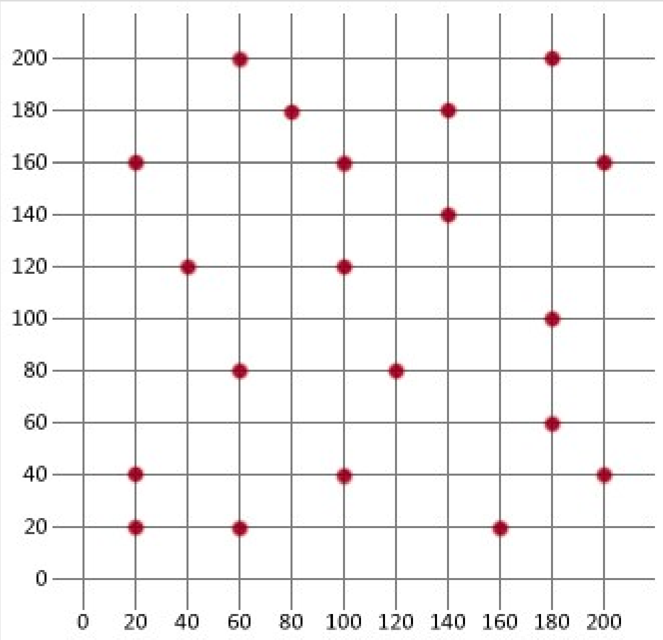
We want the temperature to be fairly high at the start (to move around the graph)

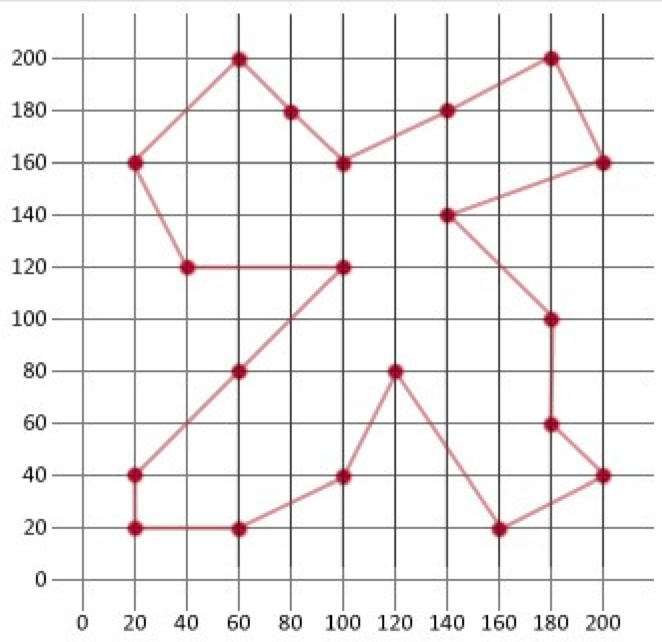
The hard part is slowly decreasing it over time

Simulated annealing

SA does work well on the traveling salesperson problem

(see: tsp.zip)





Local beam search

Beam search is similar to hill climbing, except we track multiple states simultaneously

Initialize: start with K random nodes

1. Find all children of the K nodes
2. Select best K children from whole pool
3. Repeat...

Unlike previous approaches, this uses more memory to better search "hopeful" options

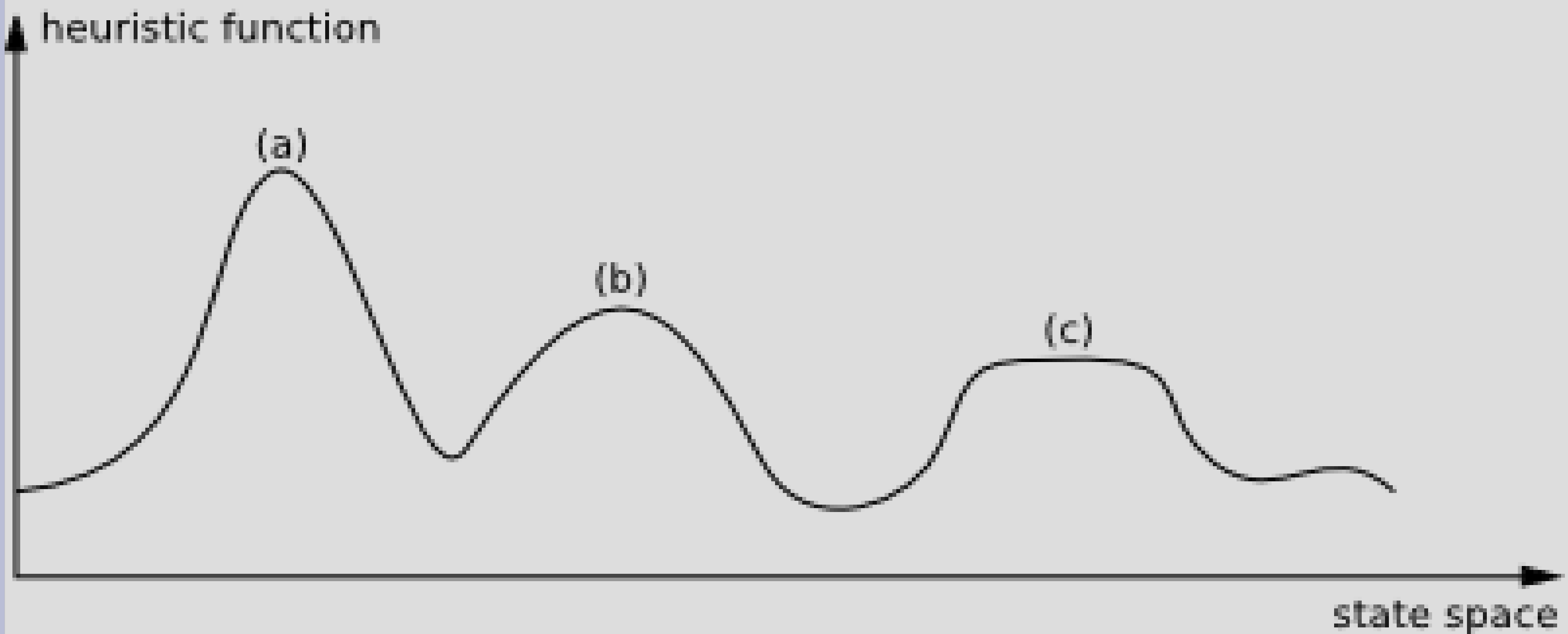
Local beam search

However, the basic version of beam search can get stuck in local maximum as well

To help avoid this, stochastic beam search picks children with probability relative to their values

This is different that hill climbing with K restarts as better options get more consideration than worse ones

Local beam search

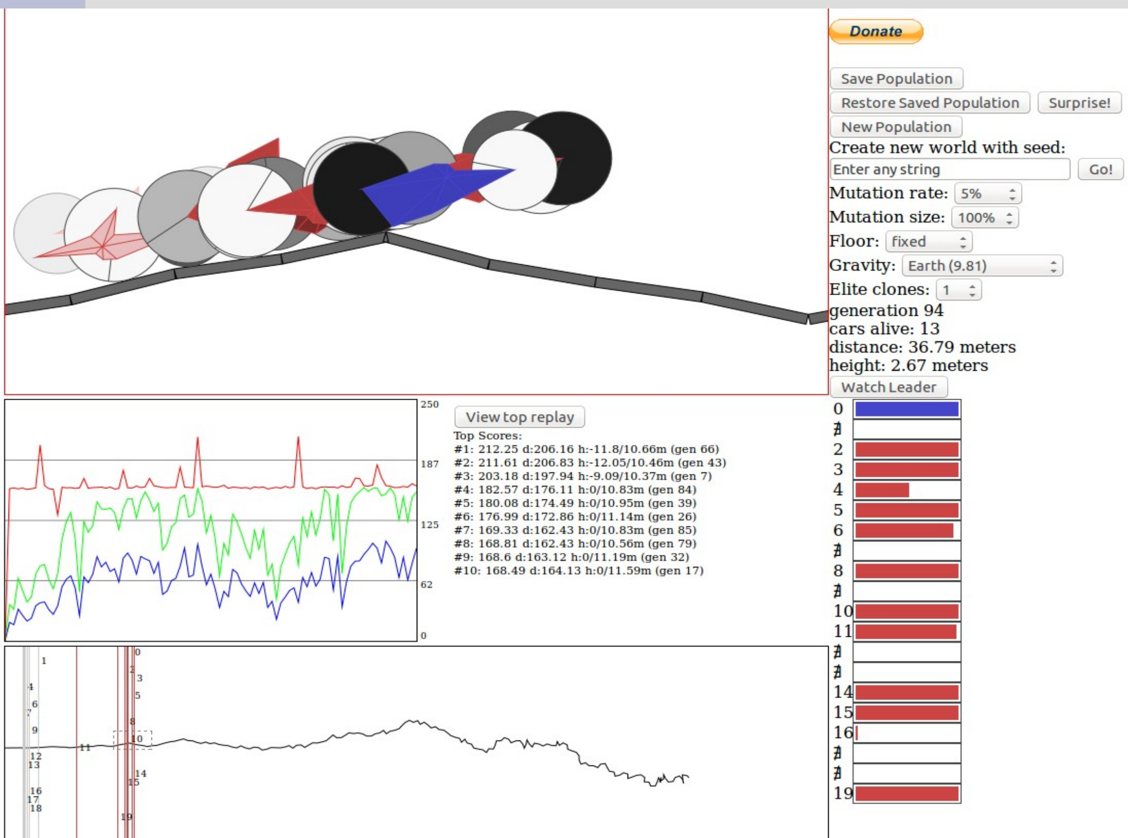


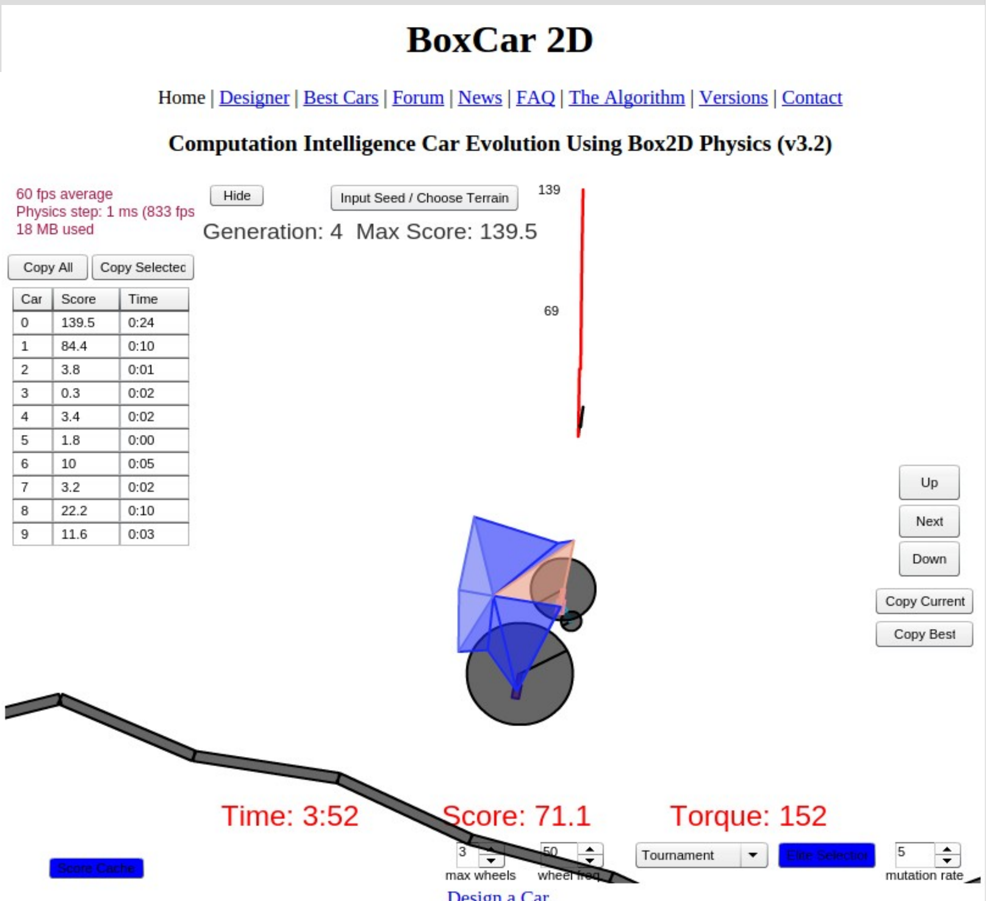
Genetic algorithms

Nice examples of GAs:

http://rednuht.org/genetic_cars_2/

http://boxcar2d.com/





Genetic algorithms

Genetic algorithms are based on how life has evolved over time

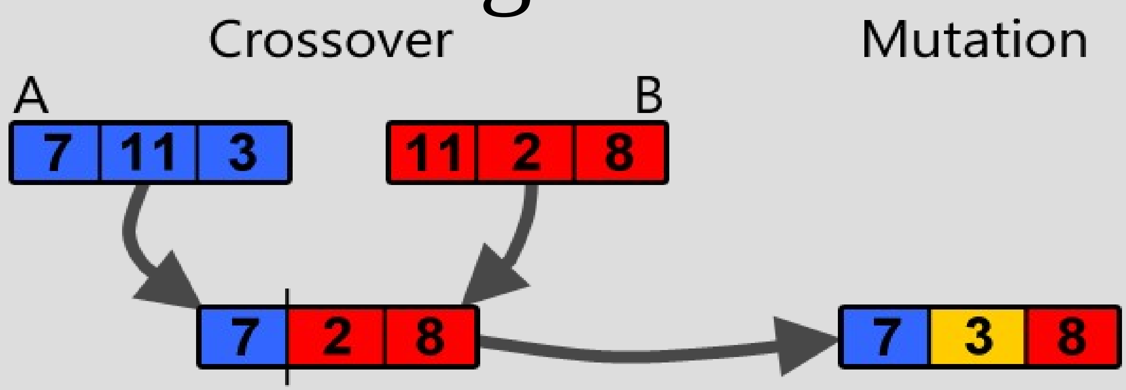
They (in general) have 3 (or 5) parts:

1. Select/generate children
    1a. Select 2 random parents
    1b. Mutate/crossover
2. Test fitness of children to see if they survive
3. Repeat until convergence

Genetic algorithms

Selection/survival:

Typically children have a probabilistic survival rate (randomness ensures genetic diversity)



Crossover:

Split the parent's information into two parts, then take part 1 from parent A and 2 from B

Mutation:

Change a random part to a random value

Genetic algorithms

Genetic algorithms are very good at optimizing the fitness evaluation function

While there are a fair amount of parameters to choose from, they are not very sensitive

The downside is that it typically takes a while to converge to the optimal solution (i.e. many generations have to be created)