

Search

additional issues

a few more approaches

Heuristics

- For A^* , the more accurate the heuristic, the better the performance.
 - fewer state will be expanded.
- Sometimes we need to be creative in developing heuristics:
 - heuristic based on a *relaxed* problem.
 - pattern databases
 - store solution to subproblems.
 - learning from experience
 - store solutions to sample problems.

OnLine Search

- Dynamic environments, things can change each time a step is made.
 - imagine solving the water jug problem when after each step someone can mess with the jugs...
- Search is *much* harder.
 - some environments can involve dead ends with no hope of backtracking.
 - actions are made, not simulated.

Online Search Strategies

- Assuming steps are reversible, DFS often makes more sense than BFS.
 - here *expanding a node* means something like "moving to a new place". Local order is obviously better than jumping from place to place.
 - Often need to keep track of all *paths* taken (not just all nodes visited).
 - Sometimes need to build a *map* of the environment.

Evolutionary Computation

- Genetic Algorithms
- Evolutionary Strategies
- Genetic Programming
- Lots of others...

Genetic Algorithms

- Find some way to encode all possible solutions (paths, parameters, whatever...) as a sequence of bits.
 - typically, fixed length sequence of bits.
 - sometimes higher-level encodings are used.
- Generate a random *population* of potential solutions.
- Evaluate each potential solution (determine the *fitness*).
- Use *fitness* to determine who is allowed to participate in reproduction.

Fitness

- Given an encoding of a possible solution, we need a function/simulation that produces a number.
 - the higher the number, the more fit the individual.
- This is the expensive part!
 - applying genetic operators is not computationally expensive.

Reproduction is blind

- Traditionally, the algorithm knows nothing about what makes an individual fit.
 - the fitness function is a black box.
 - the only comparison made is based on fitness.
- The probability of contributing to the next generation is based on fitness.
 - many schemes for this:
 - roulette wheel
 - rank ordering

Reproduction Operators

- Crossover: combine the genetic material (bits) from two *fit* individuals.
 - many variations: one point crossover, two point crossover, uniform crossover, ...
- Mutation: randomly flip bits in the poor, helpless offspring.
- Crossover *exploits* fitness, Mutation provides *exploration* (to avoid getting stuck in local optima).

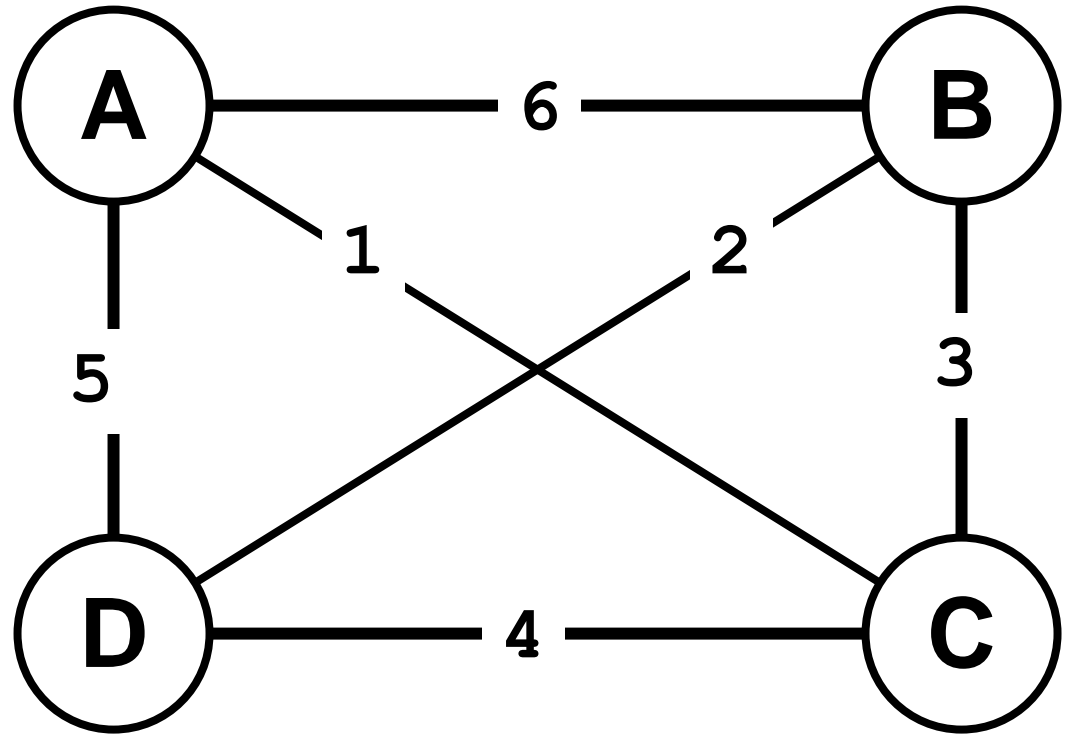
Mutation

- Pick a bit in a child randomly.
- Flip the bit

011001 => 010001

- *Rate* of mutation determines how likely each bit will be flipped.
- Sometimes the rate is decreased over time (idea borrowed from simulated annealing).

GA Example: TSP



We need a way to encode all possible solutions
tours that start at and end at A.

Options:

Clever encoding

Penalties for invalid individuals.

lots of issues here...

Evolutionary Strategies

- Like GA, but no crossover
 - Individual are selected for next generation according to fitness.
 - Only operator is mutation.
 - random search with memory.

Genetic Programming

- Potential solutions (individuals in the population) are actually computer programs.
- fitness is determined by running the programs.
- encoding is typically tree-based representation of a program.
 - LISP expressions!

GP example

A: (+ (* x y) y)

B: (- x (+ y y))

A + B =>

(+ (* x y) (+ y y)),
(- x y)