Local and Global Optimization, Online Search

Russell and Norvig 4.6

Simulated Annealing

- Physical systems are good at finding minimum energy configurations: a physical system cooled down to absolute zero will settle into its minimum energy configuration.
- Mimic the process using a probabilistic process

Simulated Annealing

function SIMULATED-ANNEALING (problem, schedule) return a solution state
input: problem, a problem
schedule, a mapping from time to temperature

initial = MAKE-NODE(INITIAL-STATE(problem))
for t ← 1 to ∞ do
    T ← schedule[t] # the system stays some time at a given temperature
    if T = 0 then return current.
    next ← a randomly selected successor of current;
    ∆E ← VALUE[current] - VALUE[next];
    if ∆E > 0 then current ← next;
    else current ← next with probability $e^{-\Delta E / T}$.

Temperature controls the probability of increasing steps.

Properties of Simulated Annealing

- Theory of Markov chains: As number of moves goes to infinity, the probability that state is some value $a$ becomes proportional to $\exp(-E/T)$.
- If temperature is lowered slowly enough - global optimum will be found with high probability. A lot of research into what makes a good cooling schedule.
- Proposed in 1983 by Scott Kirkpatrick.
- Widely used in VLSI layout, airline scheduling, etc.

Beam Search

- Variant of hill climbing:
  - Initially: $k$ random states
  - Next: determine all successors of $k$ states
  - If any of successors is optimal → done
  - Else select $k$ best from successors and repeat.
- Major difference with random-restart search:
  - Information is shared among $k$ search threads.
- Can suffer from lack of diversity.

Genetic algorithms

- Keep a population of solutions that undergo recombination and mutation
Genetic Algorithms

The state is the genetic material that makes an individual chromosome.

```
GENETIC_ALGORITHM(population, FITNESS-FN) return an individual

input: population, a set of individuals
FITNESS-FN, a function quantifying the quality of an individual

repeat
  new_population ← empty set
  loop for i from 1 to SIZE(population) do
    x ← RANDOM_SELECTION(population, FITNESS_FN)
    y ← RANDOM_SELECTION(population, FITNESS_FN)
    child ← REPRODUCE(x, y)
    MUTATE(child)
    add child to new_population
  population ← new_population
until some individual is fit enough or enough time has elapsed
return the best individual
```

Solving TSP

- Represent a tour as a permutation \((i_1, \ldots, i_n)\) of \(\{1, 2, \ldots, n\}\).
- Fitness of a solution: negative of the cost of the tour.
- Initialization: using either some heuristic, or a random set of permutations.
- Need to define crossover and mutation operations.

Crossover (2)

- **Partially Mapped (PMX) crossover:** choose a subsequence of a tour from one parent and preserve the order and position of as many cities as possible from the other parent.

PMX crossover

\(p_1 = (1 \ 2 \ 3 | 4 \ 5 \ 6 \ 7 | 8 \ 9)\)
\(p_2 = (4 \ 5 \ 2 | 1 \ 8 \ 7 \ 6 | 9 \ 3)\)
\(c_1 = (x \ x \ 2 | 4 \ 5 \ 6 \ 7 | x \ x)\)
\(c_2 = (x \ x \ x | 1 \ 8 \ 7 \ 6 | x \ x)\)

Swap defines a mapping:

\(1 \leftrightarrow 4, 6 \leftrightarrow 8, 2 \leftrightarrow 7, 5 \leftrightarrow 3, 9 \leftrightarrow 2, 8 \leftrightarrow 7, 7 \leftrightarrow 6, 6 \leftrightarrow 7.\)

The easy ones:
\(c_1 = (x \ 2 \ 3 | 1 \ 8 \ 7 \ 6 | 9 \ 2)\)
\(c_2 = (x \ x \ 2 | 4 \ 5 \ 6 \ 7 | 9 \ 3)\)

For the rest, use the mapping:
\(c_1 = (4 \ 2 \ 3 | 1 \ 8 \ 7 \ 6 | 5 \ 9)\)
\(c_2 = (1 \ 8 \ 2 | 4 \ 5 \ 6 \ 7 | 9 \ 3)\).
Mutation

Can use a 2-opt operation:
Select two points along the permutation, cut it at these points and re-insert the reversed string.
Example:
\[(1 \ 2 \ | \ 3 \ 4 \ 5 \ 6 \ | \ 7 \ 8 \ 9) \rightarrow (1 \ 2 \ | \ 6 \ 5 \ 4 \ 3 \ | \ 7 \ 8 \ 9)\]

Search in Continuous Spaces

- Example: want to place a new airport in a region, such that the sum of squared distances from each city is minimized (denoted by f).
- Can solve by:
  \[\text{gradient}(f) = 0\]
  When a closed form solution is not available (gradient ascent/descent):
  \[x \leftarrow x + \alpha \text{gradient}(f(x))\]
  How to choose \(\alpha\)? E.g. line-search

Online search

- So far, we have assumed deterministic actions and fully-known environments
  - Permits off-line search
- Consider a new problem:
  - A robot is placed in the middle of a maze
  - The task is to find the exit
- Questions: Can the robot do A* search? Can it do local search?

Formalizing online search

- Online search problems are defined by:
  - ACTIONS(s): returns the set of actions \{a_1, \ldots, a_n\} that can be executed in state s.
  - Every action returns a new state given a current state, but you have to execute it to find out what that state looks like.
  - \(C(s, a, s')\): The step-cost function tells you the cost of an action (once \(s'\) is known).
  - GOAL-TEST(s): returns true if s is a goal state.

Simple online search

- Let us assume that actions are reversible and deterministic
  - Like the robot maze (with no holes, slides, etc.)
- Then we can perform local search with an evaluation heuristic:
  - For all actions in state s:
    Try the action, leading to state s'
    Evaluate s'
    Reverse the action, returning to state s
    Select best action, going to state s'
    If s' is a goal, return; else repeat loop.

Online agents

- Difference from offline agents: An online agent can only expand the node it is physically in.
- Therefore agent needs to work locally: Online DFS, IDS.
- Possible only when actions are reversible.
The knapsack problem