Lecture 4: Search for Optimization Problems

- What is an optimization problem?
- Local search algorithms:
  - Hill climbing
  - Simulated annealing

Optimization problems

- There is some combinatorial structure to the problem
- Constraints may have to be satisfied
- But there is also a cost function, which we want to optimize!

- Or at least, we want a “good” solution
- Searching all possible solutions is infeasible
Canonical example: Traveling Salesman Problem (TSP)

- Given: a set of vertices and the distances between each pair of vertices
- Goal: construct the *shortest path that touches every vertex exactly once*
- A path that touches every vertex exactly once is called a tour.
- In the example above, \( X_1 \) is a tour, but not the optimal tour.

Real-life examples of optimization problems

- **Scheduling**
  - Given: a set of tasks to be completed, with durations and with mutual constraints (e.g. task ordering; joint resources)
  - Goal: generate the shortest schedule (assignment of start times to tasks) possible
- **VLSI circuit layout**
  - Given: a board, components and connections
  - Goal: place each component on the board such as to maximize energy efficiency, minimize production cost...
- **In AI: learning, e.g.**
  - Given: customers described by their characteristics (age, occupation, gender, location, etc) and their previous book purchases
  - Goal: find a function from customer characteristics to books which maximizes the probability of purchase
Characteristics of optimization problems

- Problem is described by a set of states (configurations) and an evaluation function
  
  E.g. in TSP, a tour is a state, and the length of the tour is the evaluation function (to minimize)

- The state space is too big to enumerate all states (or the evaluation may be expensive to compute for all states)
  
  E.g. in TSP, the state space is \((n - 1)!/2\), where \(n\) is the number of vertices to connect

- We are only interested in the best solution, not the path to the solution (unlike in \(A^*\))

- Often it is easy to find some solution to the problem

- Often it is provably very hard (NP-complete) to find the best solution

Types of search methods

1. **Constructive methods**: Start from scratch, build up a solution
   
   E.g. In TSP, start at the start city and add cities until a complete tour is formed

2. **Iterative improvement/repair methods**: Start with a solution (which may be “broken” or suboptimal) and improve it
   
   E.g. In TSP, start with a complete tour, and keep swapping cities to improve the cost

   In both cases, the search is local: we have just one solution in mind, and we look for alternatives in the "vicinity" of that solution

3. **Global search**: Start from multiple states that are far apart, and go all around the state space
Local search generic algorithm

1. Start from an initial configuration $X_0$
2. Repeat until satisfied:
   (a) Generate the set of neighbors of $X_i$ and evaluate them
   (b) Select one of the neighbors, $X_{i+1}$
   (c) The selected neighbor becomes the current configuration

Choosing well the highlighted elements is crucial for a good algorithm!

Example

```
Global optimum
Eval(X') ≥ Eval(X)
for all Xs

Local optimum
Eval(X') ≥ Eval(X)
for all Xs in Neighbors(X)
```

\[
S = \{1, \ldots, 100\}
\]

\[
Neighbors(X) = \{X-1, X+1\}
\]

How should we move around between solutions?
Hill climbing (greedy local search, gradient ascent/descent)

1. Start at initial configuration $X$ and let $E$ be the value of $X$ (high is good)
2. Repeat
   (a) Let $X_i, i = 1 \ldots n$ be the set of neighboring configurations and $E_i$ be the corresponding values
   (b) Let $E_{max} = \max_i E_i$ be the value of the best successor configuration and $i_{max} = \arg \max_i E_i$ be the index of the best configuration.
   (c) If $E_{max} \leq E$, return $X$ (we are at a local optimum)
   (d) Else let $X \leftarrow X_{i_{max}}$ and $E \leftarrow E_{max}$

Good things about hill climbing

- Trivial to program!
- Requires no memory of where we’ve been (because it does no backtracking)
- It is important to have a “good” set of neighbors (not too many, not too few)
Example: TSP, swapping two nodes

\[ O(n^2) \] comes from the fact that we have \( n \) edges in a tour, and choose two of them to swap, so there are \( \binom{n}{2} \) possible next tours.

Example: TSP, swapping three nodes

There are \( \binom{n}{3} \) combinations of edges to choose, and for each set of edges, more than one possible neighbor.
Neighborhood trade-off

- A smaller neighborhood means fewer neighbors to evaluate (so cheaper computation, but possibly worse solutions)
- A bigger neighborhood means more computation, but maybe fewer local optima, so better final result
- Defining the set of neighbors is a design choice (like choosing the heuristic for $A^*$) and has a crucial impact on performance
- For realistic problems, there may not be a unique way of defining the neighbors

Problems with hill climbing

- Can get stuck in a local maximum
- Can get stuck on a plateau
- Relies very heavily on having a good neighborhood function and a good evaluation function, in order to get an easy-to-navigate “solution landscape”
Improvements to hill climbing

- Quick fix: when stuck in a plateau or local optimum, use random restarts
- Better fix: Instead of picking the best move pick any move that produces an improvement

This is called randomized hill climbing
- But sometimes we may really need to pick apparently bad moves!

E.g. Assuming salary is the evaluation function, you can pick a dead-end job but which pays well right away, vs. picking a job that pays less now, but you learn skills that may lead to a better job later

Simulated annealing

- Allows some apparently "bad moves", in the hope of escaping local maxima
- Decrease the size and frequency of "bad moves" over time
- Algorithm sketch
  1. Start at initial configuration $X$ of value $E$ (high is good)
  2. Repeat:
    (a) Let $X_i$ be a random neighbor of $X$ and $E_i$ be its value
    (b) If $E < E_i$ then let $X \leftarrow X_i$ and $E \leftarrow E_i$
    (c) Else, with some probability $p$, still accept the move: $X \leftarrow X_i$ and $E \leftarrow E_i$
- Best solution ever found is always remembered
What value should we use for $p$?

- Suppose you are at a state of value $E$ and are considering a move to a state of lower value $E'$
- If $E - E'$ is large, you are likely close to a promising maximum, so you should be less likely to want to go downhill
- If $E - E'$ is small, the closest maximum may be shallow, so going downhill is not as bad
- We may want different neighbors with similar value to be equally likely to be picked
- As we get more experience with the problem, we may want to settle on the solution (landscape has been explored enough)

Selecting moves in simulated annealing

- If the new value $E_i$ is better than the old value $E$, move to $X_i$
- If the new value is worse ($E_i < E$) then move to the neighboring solution with probability:

$$
\exp\left(-\frac{E - E_i}{T}\right)
$$

This is called the Boltzmann distribution

- $T > 0$ is a parameter called temperature, which typically starts high, then decreases over time towards 0
- If $T$ is high, exponent is close to 0 and probability of accepting any move is close to 1
- If $T$ is very close to 0, the probability of moving to a worse solution is almost 0.
- We can decrease $T$ by multiplying with a constant $\alpha < 1$ on every move (or some other, fancier “schedule”)
Where does the Boltzmann distribution come from?

- For a solid, at temperature $T$, the probability of moving between two states of energy difference $\Delta E$ is:

$$e^{-\Delta E/kT}$$

- If temperature decreases slowly, it will reach an equilibrium, at which the probability of being in a state of energy $E$ is proportional to:

$$e^{-E/kT}$$

- So states of low energy (relative to $T$) are more likely
- In our case, states with better value will be more likely

Properties of simulated annealing

- When $T$ is high, the algorithm is in an exploratory phase (even bad moves have a high chance of being picked)
- When $T$ is low, the algorithm is in an exploitation phase (the “bad” moves have very low probability)
- If $T$ is decreased slowly enough, simulated annealing is guaranteed to reach the best solution in the limit (but there is no guarantee how fast...)
Example

![Graphs showing temperature changes over iterations for a TSP example.]

Starting point: We move most of the time uphill.

Iteration 150: Random downhill moves allow us to escape the local extremum.

TSP example

N = 13 nodes (in a circle)

Repeat K = 100N times

Optimal configuration has E = 25

Starting configuration has E = 55
**TSP example: Configurations**

The initial configuration is bottom right, final one is top left.

**TSP example: Energy**

Note that larger deviations from downhill search are allowed at high temperature.
Simulated annealing in practice

- Very useful algorithm, used to solve very hard optimization problems:
  - E.g. What gene network configuration best explains observed expression data
  - E.g. Scheduling large transportation fleets
- The temperature annealing schedule is crucial (so it needs to be tweaked)
  - Cool too fast and you do not reach optimality
  - Slow cooling leads to very slow improvements
- On large problems, simulated annealing can take days or weeks
- Simulated annealing is an example of a randomized search or Monte Carlo search
- Basic idea: run around through the environment and explore it, instead of systematically sweeping
- Very powerful for large domains!

Summary

- Optimization problems are widespread and important
- We are only interested in the final result, rather than the path to it
- It is unfeasible to enumerate all possible solutions
- Instead we can do a local search and move in the most promising direction:
  - Hill climbing (a.k.a. gradient ascent/descent) always moves in the (locally) best direction
  - Simulated annealing allows moves downhill
- Next time: global search, looking for solutions from multiple points in parallel
  - Genetic algorithms use an evolutionary-inspired procedure
  - Ant-colony optimization and other methods are also possible.
- Important lesson: the power of randomness!
  This is a key ingredient for escaping local optima