Local Search Algorithms

Chapter 4
Outline

- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms
- Ant Colony Optimization
- Tabu Search
Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it.
- Very memory efficient (only remember current state)
Example: $n$-queens

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Note that a state cannot be an incomplete configuration with $m < n$ queens.
Hill-climbing search: 8-queens problem

- $h =$ number of pairs of queens that are attacking each other, either directly or indirectly ($h = 17$ for the above state)

Each number indicates $h$ if we move a queen in its corresponding column.
Hill-climbing search: 8-queens problem

- A local minimum with $h = 1$

  what can you do to get out of this local minima?)
Hill-climbing in Continuous Spaces

- Problem: depending on initial state, can get stuck in local maxima
Gradient Descent

- Assume we have some cost-function: $C(x_1, \ldots, x_n)$
  and we want to minimize over continuous variables $x_1, x_2, \ldots, x_n$

1. Compute the gradient: $\frac{\partial}{\partial x_i} C(x_1, \ldots, x_n) \quad \forall i$

2. Take a small step downhill in the direction of the gradient:

   $$x_i \rightarrow x'_i = x_i - \lambda \frac{\partial}{\partial x_i} C(x_1, \ldots, x_n) \quad \forall i$$

3. Check if $C(x_1, \ldots, x'_i, \ldots, x_n) < C(x_1, \ldots, x_i, \ldots, x_n)$

4. If true then accept move, if not reject.

5. Repeat.
Exercise

- Describe the gradient descent algorithm for the cost function:

\[ C(x, y) = \sqrt{(x - a)^2 + (y - b)^2} \]
Line Search

- In GD you need to choose a step-size.
- Line search picks a direction, \( v \), (say the gradient direction) and searches along that direction for the optimal step:

\[
\eta^* = \arg\min C(x_t + \eta v_t)
\]

- Repeated doubling can be used to effectively search for the optimal step:

\[
\eta \rightarrow 2\eta \rightarrow 4\eta \rightarrow 8\eta \quad \text{(until cost increases)}
\]

- There are many methods to pick search direction \( v \). Very good method is “conjugate gradients”. 
Newton’s Method

- Want to find the roots of $f(x)$.

- To do that, we compute the tangent at $X_n$ and compute where it crosses the $x$-axis.

  $$\nabla f(x_n) = \frac{f(x_n) - 0}{x_{n+1} - x_n} \implies x_{n+1} = x_n - \frac{f(x_n)}{\nabla f(x_n)}$$

- Optimization: find roots of $\nabla f(x_n)$

  $$\nabla \nabla f(x_n) = \frac{\nabla f(x_n) - 0}{x_{n+1} - x_n} \implies x_{n+1} = x_n - \left[\nabla \nabla f(x_n)\right]^{-1} \nabla f(x_n)$$

- Does not always converge & sometimes unstable.

- If it converges, it converges very fast.

Basins of attraction for $x^5 - 1 = 0$; darker means more iterations to converge.
Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency.

- This is like smoothing the cost landscape.
Properties of simulated annealing search

- One can prove: If $T$ decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1 (however, this may take VERY long)

- Widely used in VLSI layout, airline scheduling, etc.
Tabu Search

• A simple local search but with a memory.

• Recently visited states are added to a tabu-list and are temporarily excluded from being visited again.

• This way, the solver moves away from already explored regions and (in principle) avoids getting stuck in local minima.
Local beam search

- Keep track of $k$ states rather than just one.
- Start with $k$ randomly generated states.
- At each iteration, all the successors of all $k$ states are generated.
- If any one is a goal state, stop; else select the $k$ best successors from the complete list and repeat.
Genetic algorithms

- A successor state is generated by combining two parent states
- Start with $k$ randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation
Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 \times 7/2 = 28)

- P(child) = 24/(24+23+20+11) = 31%
- P(child) = 23/(24+23+20+11) = 29% etc
Ant Colony Optimization

- Ant colony acts as a multi-agent system where the agents cooperate.
- Ants need to find short paths to food sources.
- Consider the travelling salesman problem.
- Construct “solutions” by joining city-to-cite steps.

Inference:

At each step an ant picks from its allowed set of (leftover) cities according to:

\[ p_{i \rightarrow j} = \frac{\gamma q_i^\alpha}{\sum_{j \in \{allowed\ \text{cities}\}} q_j^\alpha} \quad (\text{e.g. } \gamma = 1/d_{ij}) \]

- Learning: pheromone forgetting: \( q_{ij} \leftarrow (1 - \rho)q_{ij} \)
- Pheromone strengthening: \( q_{ij} \leftarrow q_{ij} + \rho \Delta_{ij} \) \( (\text{e.g. } \Delta_{ij} = a \ (\text{Length Solution})^{-\beta}) \)
Linear Programming

Problems of the sort:

\[
\begin{align*}
\text{maximize} & \quad c^T x \\
\text{subject to:} & \quad Ax \leq b; \ Bx = c
\end{align*}
\]

- Very efficient “off-the-shelves” solvers are available for LRs.
- They can solve large problems with thousands of variables.