Local Search

Search so far..

- A*, BFS, DFS etc
  - Given set of states, get to goal state

\[
\begin{array}{ccc}
7 & 2 & 4 \\
5 & 6 & \\
8 & 3 & 1 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8 \\
\end{array}
\]
  - Need to know the path as well

Example: n-queens

- Put n queens on an \(n \times n\) board with no two queens on the same row, column, or diagonal
  - How would you represent the state space of this problem?
  - How is the problem different from the 8-puzzle?

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
  - Assume access to a function, \(\text{Eval}(x)\) that tells you how good \(X\) is
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

```
function Hill-Climbing(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
  neighbor, a node
  current = Make-Node([initial state][problem])
  keep doing
    neighbor = a highest-valued successor of current
    if Value(neighbor) ≤ Value(current) then return State(current)
    current = neighbor
```

Hill-climbing search: 8-queens problem

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

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

- A local minimum with $h = 1$
Simulated annealing search

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

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
- Widely used in VLSI layout, airline scheduling, etc

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

- Variant of local beam search with sexual recombination.
Genetic Algorithms

- View optimization by analogy with evolutionary theory → Simulation of natural selection
- View configurations as individuals in a population
- View Eval as a measure of fitness
- Let the least-fit individuals die off without reproducing
- Allow individuals to reproduce with the best-fit ones selected more often
- Each generation should be overall better fit (higher value of Eval) than the previous one
- If we wait long enough the population should evolve so toward individuals with high fitness (i.e., maximum of Eval)

Genetic Algorithms: Implementation

- Configurations represented by strings:
  \[ X = 1001001 \]
- Analogy:
  - The string is the chromosome representing the individual
  - String made up of genes
  - Configuration of genes are passed on to offsprings
  - Configurations of genes that contribute to high fitness tend to survive in the population
- Start with a random population of \( P \) configurations and apply two operations
  - Reproduction: Choose 2 “parents” and produce 2 “offsprings”
  - Mutation: Choose a random entry in one (randomly selected) configuration and change it

Genetic Algorithms: Reproduction

Parents: 
\[
\begin{align*}
  & 10011001 \\
  & 10110001 \\
\end{align*}
\]

Parents: 
\[
\begin{align*}
  & 10011001 \\
  & 10110001 \\
\end{align*}
\]

Select random crossover point: 
\[
\begin{align*}
  & 10011001 \\
  & 10110001 \\
\end{align*}
\]
**Genetic Algorithms: Reproduction**

Parents:

```
0 0 1 1 0 1
1 0 1 1 0 0 1
```

Select random crossover point:

```
0 0 1 1 0 0 1
1 0 1 1 0 0 1
```

Offspring:

```
0 0 1 0 0 0 1
1 0 1 1 0 0 1
```

- Each offspring receives part of the genes from each of the parents
- Implemented by crossover operation

**Genetic Algorithms: Mutation**

- Random change of one element in one configuration
  
  - Implements random deviations from inherited traits
  
  - Corresponds loosely to "random walk": Introduce random moves to avoid small local extrema

```
1 0 0 1 1 0 1
1 0 0 1 0 0 1
1 1 1 1 0 0 1
1 1 0 1 0 0 1
1 1 1 1 0 0 1
```

- Select a random individual
- Select a random entry
- Change that entry

**Basic GA Outline**

- Create initial population \( X = \{X_1, \ldots, X_P\} \)
- Iterate:
  
  1. Select \( K \)-random pairs of parents
  2. For each pair of parents
    
    1.1 Generate offsprings \((Y_1, Y_2)\) using crossover operation
    
    1.2 For each offspring \(Y_i\):
      
      - Replace randomly selected element of the population by \(Y_i\)
      
      - With probability \(\mu\):
        
        - Apply a random mutation to \(Y_i\)
    
- Return the best individual in the population

**Genetic Algorithms: Selection**

- Discard the least-fit individuals through threshold on \(\text{Eval}\) or fixed percentage of population
- Select best-fit (larger \(\text{Eval}\)) parents in priority
- Example: Random selection of individual based on the probability distribution

\[
\Pr(\text{individual } X \text{ selected}) = \frac{\text{Eval}(X)}{\sum_{Y \in \text{population}} \text{Eval}(Y)}
\]

- Example (tournament): Select a random small subset of the population and select the best-fit individual as a parent

- Implements "survival of the fittest"
- Corresponds loosely to the greedy part of hill-climbing (we try to move uphill)
GA and Hill Climbing

- Create initial population $X = \{X_1, \ldots, X_P\}$
- Iterate:
  1. Select $K$ random pairs of parents $(X, X')$
  2. For each pair of parents $(X, X')$:
     1.1 Generate offsprings $(Y_1, Y_2)$ using crossover operation
     1.2 For each offspring $Y_i$:
         - Replace randomly selected element of the population by $Y_i$ with probability $\mu$.
         - Apply a random mutation to $Y_i$.

- Return the best individual in the population

Hill-climbing component: Try to move uphill as much as possible

Random walk component: Move randomly to escape shallow local maxima

With probability $\mu$:
- Apply a random mutation to $Y_i$

How would you set up these problems to use GA search?

TSP

SAT

$B \lor D \lor \neg E$

$\neg C \lor \neg D \lor \neg E$

$\neg A \lor \neg C \lor E$

N-Queens

GA for N-queens

TSP Example
Initial population

Best (lowest cost) element in population

Best element in population candidate for reproduction

Population at generation 15

Population at generation 35

Another TSP Example

Minimum cost

Stabilizes at generation 23

Converges and remains stable after generation 23

0.4% difference:
GA = 11.801
SA = 11.751

But: Number of operations (number of cost evaluations) much smaller for GA (approx. 2500)
GA Discussion

- Many parameters to tweak: $\mu$, $P$, $r$
- Many variations on basic scheme. Examples:
  - Multiple-point crossover
  - Dynamic encoding
  - Selection based on rank or relative fitness to least fit individual
  - Multiple fitness functions
  - Combine with a local optimizer (for example, local hill-climbing) → Deviates from “pure” evolutionary view
- In many problems, assuming correct choice of parameters, can be surprisingly effective

GA Discussion

- Why does it work at all?
- Limited theoretical results (informally!):
  - Suppose that there exists a partial assignment of genes $s$ such that:
    \[
    \text{Average of } \text{Eval}(X) \geq \text{Average of } \text{Eval}(Y)
    \]
    \[
    \text{if } s \in X \text{ and } s \not\in Y \text{ and } Y \in \text{Population}
    \]
    - Then the number of individuals containing $s$ will increase in the next generation
- Key consequence: The design of the representation (the chromosomes) is critical to the performance the GA. It is probably more important than the choice of parameters of selection strategy, etc.