Local Beam Search

Travelling Salesman Problem

1

Keeps track of k states rather than just 1. k=2 in this example. Start with k randomly generated states.

Local Beam Search Example

Travelling Salesman Problem (k=2)

2

Generate all successors of all the k states

None of these is a goal state so we continue

Select the best k successors from the complete list

Repeat the process until goal found
Local Beam Search

- How is this different from k random restarts in parallel?
- Random-restart search: each search runs independently of the others
- Local beam search: useful information is passed among the k parallel search threads
- Eg. One state generates good successors while the other k-1 states all generate bad successors, then the more promising states are expanded

Local Beam Search

- Disadvantage: all k states can become stuck in a small region of the state space
- To fix this, use stochastic beam search
- Stochastic beam search:
  - Doesn’t pick best k successors
  - Chooses k successors at random, with probability of choosing a given successor being an increasing function of its value

Genetic Algorithms

- Like natural selection in which an organism creates offspring according to its fitness for the environment
- Essentially a variant of stochastic beam search that combines two parent states (just like sexual reproduction)
- Over time, population contains individuals with high fitness

Definitions

- Fitness function: Evaluation function in GA terminology
- Population: k randomly generated states (called individuals)
- Individual: String over a finite alphabet

Definitions

- Selection: Pick two random individuals for reproduction
- Crossover: Mix the two parent strings at the crossover point
- Mutation: randomly change a location in an individual’s string with a small independent probability

Randomness aids in avoiding small local extrema
GA Overview

Population = Initial population
Iterate until some individual is fit enough or enough time has elapsed:
NewPopulation = Empty
For 1 to size(Population)
  Select pair of parents (P1, P2) using Selection(P, Fitness Function)
  Child C = Crossover(P1, P2)
  With small random probability, Mutate(C)
  Add C to NewPopulation
Population = NewPopulation
Return individual in Population with best Fitness Function

Lots of variants

• Variant1: Culling - individuals below a certain threshold are removed
• Variant2: Selection based on:

\[ P(X \text{ selected}) = \frac{\text{Eval}(X)}{\sum_{Y \in \text{Population}} \text{Eval}(Y)} \]

Example: 8-queens

• Fitness Function: number of nonattacking pairs of queens (28 is the value for the solution)
• Represent 8-queens state as an 8 digit string in which each digit represents position of queen

Example: 8-queens (Fitness Function)

<table>
<thead>
<tr>
<th>Initial Population</th>
<th>Fitness Function</th>
<th>Selection</th>
<th>Crossover</th>
<th>Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>24749552</td>
<td>2463214</td>
<td>24415241</td>
<td>24415241</td>
<td>24415241</td>
</tr>
<tr>
<td>32752411</td>
<td>24752411</td>
<td>24425241</td>
<td>24425241</td>
<td>24425241</td>
</tr>
<tr>
<td>32543221</td>
<td>24415241</td>
<td>24415241</td>
<td>24415241</td>
<td>24415241</td>
</tr>
<tr>
<td>32752411</td>
<td>24752411</td>
<td>24425241</td>
<td>24425241</td>
<td>24425241</td>
</tr>
</tbody>
</table>

Example: 8-queens (Fitness Function)

<table>
<thead>
<tr>
<th>Initial Population</th>
<th>Fitness Function</th>
<th>Selection</th>
<th>Crossover</th>
<th>Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>24749552</td>
<td>2463214</td>
<td>24415241</td>
<td>24415241</td>
<td>24415241</td>
</tr>
<tr>
<td>32752411</td>
<td>24752411</td>
<td>24425241</td>
<td>24425241</td>
<td>24425241</td>
</tr>
<tr>
<td>32543221</td>
<td>24415241</td>
<td>24415241</td>
<td>24415241</td>
<td>24415241</td>
</tr>
<tr>
<td>32752411</td>
<td>24752411</td>
<td>24425241</td>
<td>24425241</td>
<td>24425241</td>
</tr>
</tbody>
</table>

Values of Fitness Function

<table>
<thead>
<tr>
<th>Fitness Function</th>
<th>Probability of selection (proportional to fitness score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2463214</td>
<td>31%</td>
</tr>
<tr>
<td>24752411</td>
<td>24%</td>
</tr>
<tr>
<td>24425241</td>
<td>24%</td>
</tr>
<tr>
<td>24415241</td>
<td>14%</td>
</tr>
<tr>
<td>24415241</td>
<td>14%</td>
</tr>
</tbody>
</table>

Example: 8-queens

Values of Fitness Function
Example: 8-queens (Selection)

Notice 3 2 7 5 2 4 1 1 is selected twice while 3 2 5 4 3 2 1 3 is not selected at all.

Example: 8-queens (Crossover)

Example: 8-queens (Mutation)

Mutation corresponds to randomly selecting a queen and randomly moving it in its column.

Implementation details on Genetic Algorithms

- Initially, population is diverse and crossover produces big changes from parents.
- Over time, individuals become quite similar and crossover doesn’t produce such a big change.
- Crossover is the big advantage:
  - Preserves a big block of “genes” that have evolved independently to perform useful functions
  - Eg. Putting first 3 queens in positions 2, 4, and 6 is a useful block

Schemas

- A substring in which some of the positions can be left unspecified eg. 246****
- Instances: strings that match the schema
- If the average fitness of the instances of a schema is above the mean, then the number of instances of the schema within the population will grow over time.
The fine print…

- The representation of each state is critical to the performance of the GA
- Lots of parameters to tweak but if you get them right, GAs can work well
- Limited theoretical results (skeptics say it’s just a big hack)

And remember….

```
def getSolutionCosts(navigationCode):
    initialCost = 15
    extraComputationCost = 8

    thisAlgorithmHasThreeCosts = 99999999
    waterComputationCost = 415

    GENETIC ALGORITHMS TIP:
    ALWAYS INCLUDE THIS IN YOUR FITNESS FUNCTION

    (From http://www.xkcd.com/534/)
```

Discrete Environments

Gradient Descent

```
Hillclimbing pseudocode
X ← Initial configuration
Iterate:
    E ← Eval(X)
    N ← Neighbors(X)
    For each X_i in N
        E_i ← Eval(X_i)
        E* ← Highest E_i
        X* ← X_i with highest E_i
    If E* > E
        X ← X*
    Else
        Return X
```

Local Search in Continuous State Spaces

- Almost all real world problems involve continuous state spaces
- To perform local search in continuous state spaces, you need techniques from calculus
- The main technique to find a minimum is called gradient descent (or gradient ascent if you want to find the maximum)

Gradient Descent

- What is the gradient of a function $f(x)$?
  - Usually written as
    $$\nabla f(x) = \frac{\partial}{\partial x} f(x)$$
  - $\nabla f(x)$ (the gradient itself) represents the direction of the steepest slope
  - $|\nabla f(x)|$ (the magnitude of the gradient) tells you how big the steepest slope is
Gradient Descent
Suppose we want to find a local minimum of a function \( f(x) \). We use the gradient descent rule:
\[
x \leftarrow x - \alpha \nabla f(x)
\]
\( \alpha \) is the learning rate, which is usually a small number like 0.05.
Suppose we want to find a local maximum of a function \( f(x) \). We use the gradient ascent rule:
\[
x \leftarrow x + \alpha \nabla f(x)
\]

Question of the Day
• Why not just calculate the global optimum using \( \nabla f(x) = 0 \)?
  – May not be able to solve this equation in closed form
  – If you can’t solve it globally, you can still compute the gradient locally (like we are doing in gradient descent)

Multivariate Gradient Descent
• What happens if your function is multivariate eg. \( f(x_1, x_2, x_3) \)?
• Then
\[
\nabla f(x_1, x_2, x_3) = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right)
\]
• The gradient descent rule becomes:
\[
x_1 \leftarrow x_1 - \alpha \frac{\partial f}{\partial x_1}, \quad x_2 \leftarrow x_2 - \alpha \frac{\partial f}{\partial x_2}, \quad x_3 \leftarrow x_3 - \alpha \frac{\partial f}{\partial x_3}
\]

More About the Learning Rate
• If \( \alpha \) is too large
  – Gradient descent overshoots the optimum point
• If \( \alpha \) is too small
  – Gradient descent requires too many steps and will take a very long time to converge
Weaknesses of Gradient Descent

1. Can be very slow to converge to a local optimum, especially if the curvature in different directions is very different
2. Good results depend on the value of the learning rate $\alpha$
3. What if the function $f(x)$ isn’t differentiable at $x$?

What you should know

- Be able to formulate a problem as a Genetic Algorithm
- Understand what crossover and mutation do and why they are important
- Differences between hillclimbing, simulated annealing, local beam search, and genetic algorithms
- Understand how gradient descent works, including its strengths and weaknesses
- Understand how to derive the gradient descent rule