Advanced Search Algorithms II

CSE 415: Introduction to Artificial Intelligence
University of Washington
Winter, 2018

Genetic Search

Employ probabilistic state changes to help escape from local minima or maxima.

Combine strengths from multiple candidate solutions.

Motivation

1. Use nature as a guide. (Mutation plus mitotic reproduction, and Darwin's principle of natural selection.)

2. Take advantage of parallel processing. (Allow an entire population -- maybe millions or billions -- to evolve.)

3. Use randomness to escape from local maxima of the fitness function.

The Evolutionary Model

Goal: To produce an individual having certain characteristics. (A "most fit" individual.)

Method: Create a diverse population of individuals. Provide a means for adding to the population. Provide a means to weed out less fit individuals. Let the population evolve.

Mutation: Random small change to the genetic blueprint for an individual.

Crossover: Formation of a genetic blueprint for a new individual by splicing together a piece from individual A with a piece from individual B.

Fitness function: A function that maps each individual to a scalar fitness value.

Mutation and Crossover

Mutation: Makes a move or a perhaps a sequence of moves in the state space. If the direction of moving is random, there is a good chance to escape from local maxima.
Type 1: A random modification of one atomic unit -- one gene.
Type 2: A reordering of genes, .e.g, a transposition of two genes.

Crossover: Formation of a genetic blueprint for a new individual by splicing together a piece from individual A with a piece from individual B.

If A and B are each very fit, but in different ways, perhaps crossover will produce an even more fit individual.

G.S. Example: A Travelling Salesman Problem

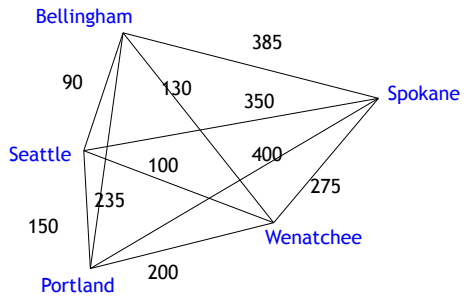
Find a shortest tour for the cities:
Seattle, Bellingham, Spokane, Wenatchee, Portland.

Given: A matrix of intercity distances for these cities.

Tour: a Hamiltonian circuit -- a closed path that starts and ends at one city and visits every other city exactly once.

Our assumption: It's possible to go from any city A to another city B without stopping (visiting) any other city C.

The Map



Problem Representation

Each individual: A list of 5 cities, repetitions permitted.
e.g., (SEATTLE, SEATTLE, SPOKANE, SPOKANE, SPOKANE)

Fitness function: $f(x) = \frac{10000}{2 f_1(x) + 50 f_2(x)}$

Path cost: $f_1(x) = \sum_{i=0}^{4} \text{dist}(x[i], x[i+1 \bmod 5])$

Non-tour penalty:
$f_2(x) = 100 (| \text{Cities} - \text{cities}(x) | + | \text{cities}(x) - \text{Cities} |)$
where "-" denotes set difference and |s| denotes cardinality.

Representation for Individuals

`[ citySequence, strengthValue]`

An initial-state individual:

```
[['Seattle', 'Seattle', 'Seattle', 'Seattle', 'Seattle'], 0.5]
```

A goal-state individual:

```
[['Seattle', 'Bellingham', 'Spokane', 'Wenatchee', 'Portland'],
4.8]
```


MUTATE1

```
def mutate1(individual):
    where = choice(range(NCITIES))
    newCity = choice(CITIES)
    newIndiv = [individual[0][:], individual[1]]
    newIndiv[0][where]=newCity
    return newIndiv
```


MUTATE2

```
def mutate2(individual):
    where1 = choice(range(NCITIES))
    where2 = choice(range(NCITIES))
    city1 = individual[0][where1]
    city2 = individual[0][where2]
    newIndiv = [individual[0][:], individual[1]]
    newIndiv[0][where1]=city2
    newIndiv[0][where2]=city1
    return newIndiv
```


CROSSOVER

```
# In this function, individual1 and individual2
# are the paths, without the strength values.
# In theory there are two new individual produced,
# but we are only returning one of them in this
# implementation.

def crossover(individual1, individual2):
    where = choice(range(NCITIES))
    newIndiv1 = individual1[:where] +\
                    individual2[where:]
    #newIndiv2 = individual2[:where] +\
    #               individual1[where:]
    return newIndiv1
```

EVOLVE

```
def evolve(ngenerations, nmutations, ncrossovers):
    global POPULATION; global INITIAL_POPULATION;
    POPULATION = INITIAL_POPULATION
    for i in range(ngenerations):
        for j in range(nmutations):
            mutant = mutate(choice(POPULATION))
            print 'mutant is ' + str(mutant)
            addIndividual(mutant)
        for j in range(ncrossovers):
            theCross = crossover(choice(POPULATION),\
                                 choice(POPULATION))
            print 'theCross is ' + str(theCross)
            addIndividual(theCross)
        print 'In generation ' + str(i) +\
              ', the population is: '
        print str(POPULATION)
```


Design Issues

Individual needs to be represented as a linear string or list to be amenable to crossovers.

Fitness function must permit some diversity in the population in order to avoid getting stuck in local maxima.

One could allow the mutation mechanisms and fitness functions to change during a run so that greater diversity is permitted near the beginning, but the search can "tighten up" later. (compare to simulated annealing).

Randomness vs deterministic choices.

Can genetic search be forced to systematically cover the entire state space?