Lecture04a: Introduction to Evolutionary Algorithms

CS540 2/6/18

Announcements

On-campus students:
- Make sure I am wearing the microphone
- We are going to keep using the 2nd microphone for questions

All students:
- No paper this week – work on your projects!
- A week from Thursday will be project presentations
  - Send me Powerpoint or pdf files by Wednesday night
- Team checkin…

Evolutionary Algorithm 0.1

(1) Create initial population
(2) While time remaining
  (a) Repeat many times to create children
    (i) Randomly select two elements from previous population
    (ii) Create "child" element by genetic crossover
    (iii) With small probability, mutate child
    (iv) add child to pool
  (b) Probabilistically select |P| children from pool based on fitness
  (c) Replace previous population with new population
(3) Select best element from final population

Note: there are a lot of variants on this theme.

Evolutionary Algorithms

Analogies
- Population genetics
- Simulated evolution

Population based
- Every member is a feasible solution
- Population as a whole models many points in search space
- Highly parallel

Decomposable states
- Feasible solutions not just "points"
- Feasible solutions have good aspects and bad aspects
- Feasible solutions can be combined to create new solutions

Computational Advantages

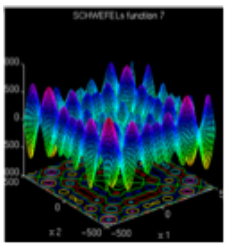
No gradient information needed
- Fitness function evaluates quality
- Fitness of neighbors never compared

Global (not local) search
- No hill climbing, no local minima

Potential for massive parallelism

Easily hybridized
- E.g. by having population elements do a local search

Visualizing Search Spaces



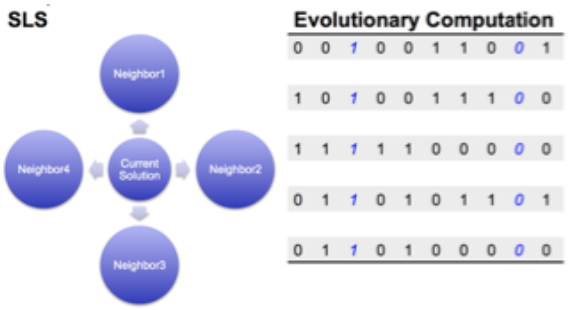
Assume (x,y) are features
Assume height is quality

Local search:
- State is a point (x,y)
- Neighborhood is a grid (usually)

Parallel local search
- State is a set of points {(x,y),…}
- Neighborhood still fixed

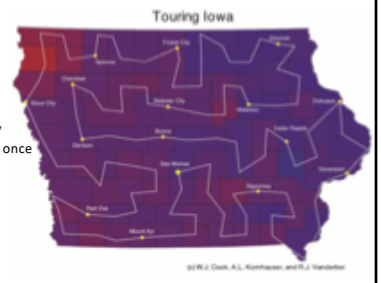
Evolutionary algorithms
- State as a set of elements
- Elements are decomposable
- No fixed neighborhood structure

Visualizing Search Spaces (II)



Example: Touring Salesman

Task:
Find a path that:
(1) Starts and ends at the same city
(2) Contains every other city eactly once
(3) Has minimum total length



Evolutionary Algorithm for TSP

Choose a representation
- Number cities from 1 to N
- Any permutation of the numbers 1 to N is a valid tour
  - Assuming wrap-around from last city back to first

Fitness function
- $\|(x_1, y_1) - (x_N - y_N)\| + \sum_{i=1}^{N-1} \|(x_i, y_i) - (x_{i+1}, y_{i+1})\|$

Initial Population
- |P| random permutations of the vector [1,…,N]

Evolutionary TSP (cont.)

Crossover Operator
- Takes two "parent" elements, produces a "child"
- Example: ordered crossover
  - Select random interval of first parent, copy it in place to child
  - Insert missing elements into child, in order they appear in second parent
    - Parent 1: [1,2,3,4,5,6,7,8,9]
    - Parent 2: [9,8,7,6,5,4,3,2,1]
    - Child: [9,5,4,3,2,6,7,8,1]

Mutation operator
- Must produce valid element, given an element
- Example: swap mutation
  - Randomly swap two elements in the tour
  - Input: [1,2,3,4,5,6,7,8,9]
  - Output: [1,2,3,7,5,6,4,8,9]

Evolutionary TSP (III)

Now pick some parameters
1. Population size
2. Children per generation
3. Number of generations

Run algorithm, pick best element of final generation

Example

Let's consider just five cities (lat, long):
- 1: Cherokee (42.7, 95.5)
- 2: Davenport (41.5, 90.6)
- 3: Des Moines (41.6, 93.6)
- 4: Dubuque (42.5, 90.7)
- 5: Sioux City (42.5, 96.4)

Pairwise distances precomputed in a table:

	1	2	3	4	5
1	0	5.0	2.2	4.8	0.9
2	5.0	0	2.1	1.0	5.9
3	2.2	2.1	0	3.0	2.9
4	4.8	1.0	3.0	0	5.7
5	0.9	5.9	2.9	5.7	0

Example (II)

Initial Population
- Chosen randomly, size 3 (fitness in red)
- {[1,3,4,2,5] 13, [4,3,5,1,2] 12.8, [2,4,5,1,3] 11.9}

Generate a child
- [1,3,4,2,5] + [2,4,5,1,3] = [2,3,4,5,1] (16.7)

Mutate child (only do this sometimes)
- [2,3,4,5,1] = [2,1,4,5,3](20.5)

Add child to pool
- {[2,1,4,5,3](20.5)}

Example (III)

Generate another child
- [4,3,5,1,2] + [1,3,4,2,5] = [3,4,5,1,2](15.6)
- Don't mutate this one (probabilistic)
- Add to pool {[2,1,4,5,3](20.5), [3,4,5,1,2](15.6)}

And another child
- [2,4,5,1,3] + [4,3,5,1,2] = [2,4,3,5,1](12.8)
- Mutate [2,4,3,5,1] = [4,2,3,5,1](11.8)

Select new population, with preference by fitness
- {[4,2,3,5,1] (11.8), [2,1,4,5,3](20.5)}