EC-350 AI and Decision Support Systems

Week 6
Local Search Algorithms

Dr. Arslan Shaukat

NUST

Acknowledgement: Lecture slides material from Stuart Russell

Local Search Algorithms

- In many problems the path to the goal is irrelevant
  - *8 queens problem, integrated circuit design, automatic prog*
  - *Factory floor layout, Telecom network optimization*
- Algorithms to solve such problems are called local search algorithm
  - *Start from the current state and then gradually only move to the neighbor of the state*
  - *Don't have to remember all the previous states*

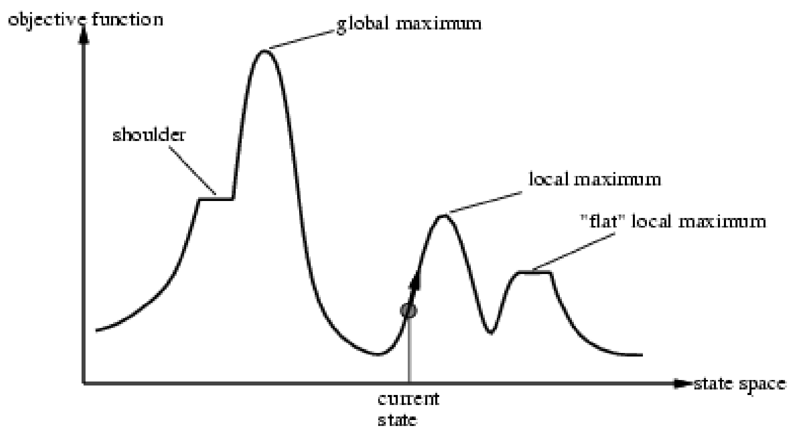
Local Search Algorithms

- They use very little memory – usually constant amount
- They can often find reasonable solution in a large or infinite space for which systematic algorithms are unsuitable
- Useful for solving pure optimization problems in which the aim is to find best state according to an objective function

Local Search Algorithms

- Consider the state space as a land scape
- Locations and elevation defined by state and value of the objective function respectively
- If elevation is a cost then aim is to find the lowest valley – a global minimum
- If elevation is objective function then aim is to find the highest peak – a global maximum
- A complete local search algorithm always finds a goal if one exist, an optimal always find a global min/max

State Space Landscape



Genetic Algorithms

- Organisms (animals or plants) produce a number of off-springs which are almost, but not entirely, like themselves
  - *Variation may be due to mutation (random changes)*
  - *Variation may be due to the fact that an offspring has some characteristics from each parent)*
- Some of these offspring may survive to produce offspring of their own—some won't
  - *The "better adapted" offspring are more likely to survive*
  - *Over time, later generations become better and better adapted*
- Genetic algorithms use this same process to "evolve" better programs.

Genes and Chromosomes

- Genes are the basic "instructions" for building an organism
- A chromosome is a sequence of genes
- Biologists distinguish between an organism's genotype (the genes and chromosomes) and its phenotype (what the organism actually is like)
  – *Example: You might have genes to be tall, but never grow to be tall for other reasons (such as poor diet)*
- Similarly, "genes" may *describe* a possible solution to a problem, without actually *being* the solution.

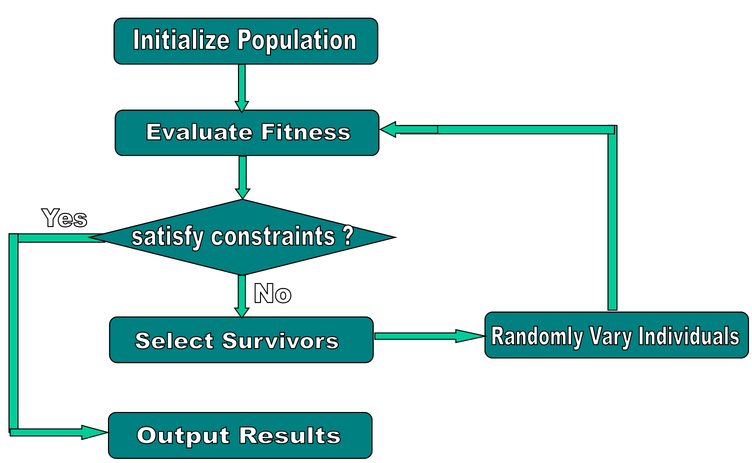
Quick Overview

- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  – *discrete optimization*
- Attributed features:
  – *Not too fast*

The Basic GA

- Start with a large "population" of randomly generated "attempted solutions" to a problem
- Repeatedly do the following:
  - *Evaluate each of the attempted solutions through a fitness function*
  - *Keep a subset of these solutions (ones with the "best" fitness)*
  - *Use these solutions to generate a new population*
- Quit when you have a satisfactory solution (or you run out of time)

Conceptually…



A Really Simple Example

- Suppose your "organisms" are 32-bit computer words
- You want a string in which all the bits are ones
- Here's how you can do it:
  - *Create 100 randomly generated computer words*
  - *Repeatedly do the following:*
    - Count the 1 bits in each word
    - Exit if any of the words have all 32 bits set to 1
    - Keep the ten words that have the most 1s (discard the rest)
    - From each word, generate 9 new words as follows:
      - *Pick a random bit in the word and toggle (change) it*
- Note that this procedure does not guarantee that the next "generation" will have more 1 bits, but it's likely.

A More Realistic Example (I)

- Suppose you have a large number of (x, y) data points
  - *For example, (1.0, 4.1), (3.1, 9.5), (-5.2, 8.6), ...*
- You would like to fit a polynomial (of up to degree 5) through these data points
  - *That is, you want a formula* $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$ *that gives you a reasonably good fit to the actual data*
  - *Here's the usual way to compute goodness of fit: Compute the sum of* $(\text{actual } y - \text{predicted } y)^2$ *for all the data points: The lowest sum of differences represents the best fit*
- There are some standard curve fitting techniques, but let's assume you don't know about them
- You can use a genetic algorithm to find a "pretty good" solution.

A More Realistic Example (II)

- Your formula is $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$
- Your "genes" are a, b, c, d, e, and f
- Your "chromosome" is the array [a, b, c, d, e, f]
- Your evaluation function for *one* array is:
  - *For every actual data point (x, y), (red means "actual data")*
    - Compute $ý = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$
    - Find the sum of $(y - ý)^2$ over all x
    - The sum is your measure of "badness" (larger numbers are worse)
  - *Example: For [0, 0, 0, 2, 3, 5] and the data points (1, 12) and (2, 22):*
    - $ý = 0x^5 + 0x^4 + 0x^3 + 2x^2 + 3x + 5$ is $2 + 3 + 5 = 10$ when x is 1
    - $ý = 0x^5 + 0x^4 + 0x^3 + 2x^2 + 3x + 5$ is $8 + 6 + 5 = 19$ when x is 2
    - $(12 - 10)^2 + (22 - 19)^2 = 2^2 + 3^2 = 13$
    - If these are the only two data points, the "badness" of [0, 0, 0, 2, 3, 5] is 13

A More Realistic Example (III)

- Your algorithm might be as follows:
  - *Create 100 six-element arrays of random numbers*
  - *Repeat 500 times (or any other number):*
    - For each of the 100 arrays, compute its badness (using all data points)
    - Keep the ten best arrays (discard the other 90)
    - From each array you keep, generate nine new arrays as follows:
      - *Pick a random element of the six*
      - *Pick a random floating-point number between 0.0 and 2.0*
      - *Multiply the random element of the array by the random floating-point number*
  - *After all 500 trials, pick the best array as your final answer.*

Generating a Population

- Through one parent:
  - *In the previous example, each solution had only one parent*
  - *The only way to introduce variation was through mutation (random changes)*
- Through two parents:
  - *Each solution has two parents*
  - *New solutions are produced by combining parts of the chromosomes of each parent – more commonly known as crossover.*

The Really Simple Example Again

- Suppose your "organisms" are 32-bit computer words, and you want a string in which all the bits are ones
- Here's how you can do it:
  - *Create 100 randomly generated computer words*
  - *Repeatedly do the following:*
    - Count the 1 bits in each word
    - Exit if any of the words have all 32 bits set to 1
    - Keep the ten words that have the most 1s (discard the rest)
    - From each word, generate 9 new words as follows:
      - *Choose one of the other words*
      - *Take the first half of this word and combine it with the second half of the other word*

The Example Continued

- Half from one, half from the other:
  0110 1001 0100 1110 1010 1101 1011 0101
  1101 0100 0101 1010 1011 0100 1010 0101
  0110 1001 0100 1110 1011 0100 1010 0101
- Or we might choose "genes" (bits) randomly:
  0110 1001 0100 1110 1010 1101 1011 0101
  1101 0100 0101 1010 1011 0100 1010 0101
  0100 0101 0100 1010 1010 1100 1011 0101
- Or we might consider a "gene" to be a larger unit:
  0110 1001 0100 1110 1010 1101 1011 0101
  1101 0100 0101 1010 1011 0100 1010 0101
  1101 1001 0101 1010 1010 1101 1010 0101

Comparison of Simple Examples

- In the simple example (trying to get all 1s):
  - *The two-parent-no mutation approach, if it succeeds, is likely to succeed much faster*
    - Because up to half of the bits change each time, not just one bit
  - *However, with no mutation, it may not succeed at all*
    - By pure bad luck, maybe *none* of the first (randomly generated) words have (say) bit 17 set to 1
      - *Then there is no way a 1 could ever occur in this position*
    - Another problem is lack of genetic diversity
      - *Maybe some of the first generation did have bit 17 set to 1, but none of them were selected for the second generation*
- The best technique *in general* turns out to be a two-parent approach with a *small* probability of mutation

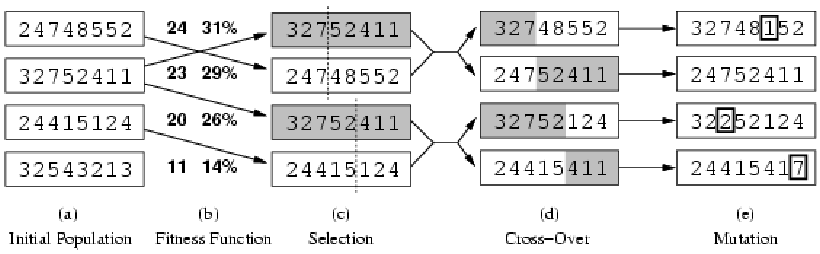
Un-Directed Evolution

- In the previous examples, child organisms were formed randomly
  - *We didn't choose the "best" gene from each parent*
  - *That's how biological evolution works: it's not necessary that a child will inherit only the best characteristics of both the mother and father.*

Genetic Algorithms

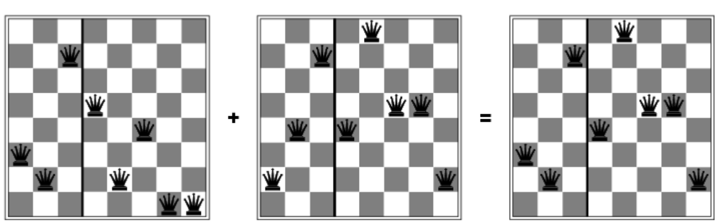
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Start with k randomly generated states (*population*)
- Evaluation function (*fitness function*). Higher values for better states.
- A successor state is generated by combining two parent states
- Produce the next generation of states by
  - *selection,*
  - *crossover, and*
  - *mutation*

8 Queens Example



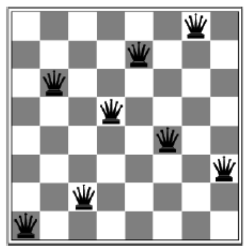
- Fitness function: number of non-attacking pairs of queens (min = 0, max = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Example



Fitness Function

- *h*: use the number of attacking pairs of queens.
- There are 28 pairs of different queens, so solutions have fitness 28. (Basically, **fitness function** is **28 − *h***)
- for example, fitness of the state below is 27 (queens in columns 4 and 7 attack each other)



8 Slider Example

- In a block slider problem, player bring blocks to their goal state, using 8 moves of the chromosome.
- **Each Move in chromosome represents movement of the blank space**. E.g.in the chromosome shown
  - *if Move 1 = Right, the blank space will come to right, and block 7 will move to left*
  - *Move 2=Up, so blank space will move up and block 8 will come down.*
  - *Move 3=Left, so blank space will move left and block 2 will move right*

8 slider problem

4	1	3
2	8	5
Blank	7	6

Problem state

→

1	2	3
4	5	6
7	8	Blank

Goal state

Chromosome structure

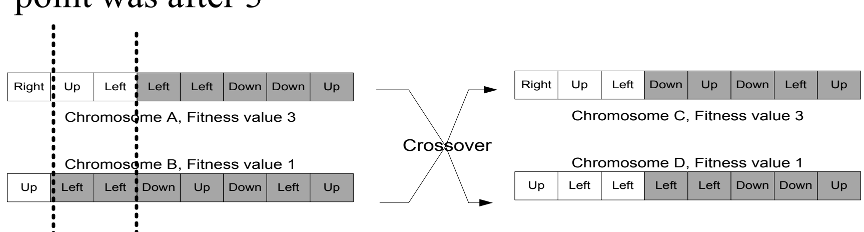
Right	Up	Left	Left	Left	Down	Down	Up
Move 1	Move 2	Move 3	Move 4	Move 5	Move 6	Move 7	Move 8

Example

- In generation1: Find fitness, and select pairs for crossover.
- In generation 2: Crossover, Find fitness, and worst chromosome will mutate, rest will remain as it is.
- In generation 3: Find fitness and select pairs for crossover.
- In generation 4: Crossover and find fitness
- **Fitness:**
  - ***The number of moves in which BLANK space can move is fitness****. E.g. If Move 1 =Up and Move 2 = Left. Then blank space will move just one space, up, and cannot go left after that. So fitness value is 1.*

Example

- **Crossover:**
- Arrange chromosome in order of fitness.
- Pair 1: 1st and 4th
- Pair2: 2nd and 3rd
- Crossover point is chosen at the maximum of both fitness values, i.e. out of fitness 1 and fitness 3, the crossover point was after 3



Example

- **Mutation:**
- For mutation, the worst chromosome changes its (first problematic) direction. i.e.
- Up changes to down, and vice versa and left changes to right and vice versa

Chromosome B, Fitness value 1

Up	Left	Left	Down	Up	Down	Left	Up

Mutated →

Chromosome, Fitness value 3

Up	Right	Left	Down	Up	Down	Left	Up

Generation A

Move 1	Move 2	Move 3	Move 4	Move 5	Move 6	Move 7	Move 8	Fitness value	Names
Up	Right	Up	Down	Down	Right	Left	Down		
Up	Left	Down	Up	Down	Up	Right	Right		
Up	Up	Right	Up	Right	Up	Right	Right		
Right	Up	Left	Up	Left	Down	Right	Down		

↓ After crossover

Generation B

Move 1	Move 2	Move 3	Move 4	Move 5	Move 6	Move 7	Move 8	Fitness value	Names

↓ After mutation

Generation C

Move 1	Move 2	Move 3	Move 4	Move 5	Move 6	Move 7	Move 8	Fitness value	Names

↓ After crossover

Generation D

Move 1	Move 2	Move 3	Move 4	Move 5	Move 6	Move 7	Move 8	Fitness value	Names

Applications

- http://brainz.org/15-real-world-applications-genetic-algorithms/

15 Real-World Uses of Genetic Algorithms

399 Like 149 Tweet 19 Pin it Comments

Genetic Algorithm: A heuristic search technique used in computing and Artificial Intelligence to find optimized solutions to search problems using techniques inspired by evolutionary biology: mutation, selection, reproduction [inheritance] and recombination.

Assignment# 2

- Finding solution with Genetic Algorithm
- Submission: After 2 weeks