Heuristics for Optimization

Olga Galinina
olga.galinina@tut.fi

ELT-53656 Network Analysis and Dimensioning II
Department of Electronics and Communications Engineering
Tampere University of Technology, Tampere, Finland

March 19, 2014
Heuristics for Optimization

Outline

1. Problem formulation
2. Genetic algorithms
3. Simulated Annealing
Outline

1 Problem formulation

2 Genetic algorithms

3 Simulated Annealing
Preamble

Branch and bound algorithms:
  guarantee that the optimum solution will be found

But!

- It is combinatorially explosive
- problems common in practice are often problems of large scale
Branch and bound algorithms: guarantee that the optimum solution will be found

But!

- It is combinatorially explosive
- problems common in practice are often problems of large scale

→ We are not looking for optimal point, but *approximate solution* within the limits of time and computer memory available
Heuristic methods

Heuristic

is a method that is not guaranteed to find the optimum, but usually gives a very good solution, though it cannot guarantee to do even that every time.

Heuristics are "quick and dirty" methods, generally relatively fast and relatively good.
Heuristic methods

Heuristic

is a method that is not guaranteed to find the optimum, but usually gives a very good solution, though it cannot guarantee to do even that every time.

Heuristics are "quick and dirty" methods, generally relatively fast and relatively good.
Heuristic methods

Heuristic

is a method that is not guaranteed to find the optimum, but usually gives a very good solution, though it cannot guarantee to do even that every time.

Heuristics are ”quick and dirty” methods, generally relatively fast and relatively good. It has minimum length over all forward paths with the same origin and destination nodes. The length is a shortest distance.
Stochastic Optimization

Refers to the minimization (or maximization) of a function in the presence of randomness in the optimization process.

The randomness:
- noise in measurements
- Monte Carlo randomness in the search procedure

Common methods of stochastic optimization:
- direct search methods (such as the Nelder-Mead method)
- stochastic approximation
- stochastic programming
- and miscellaneous methods such as simulated annealing and genetic algorithms
What is the probability to meet your friend if you agreed to meet from 13 till 14 and can wait for each other exactly 5 minutes?
Mental break

What is the probability to meet your friend if you agreed to meet from 13 till 14 and can wait for each other exactly 5 minutes?

\[ 0 \geq x, y \geq 60 \text{ - time of arrivals} \]
\[ xy < 5, y > x \]
\[ xy < 5, x > y \]

\[
P(A) = \frac{S_g}{S} = \frac{60 \cdot 60 - 55 \cdot 55}{60 \cdot 60} = \frac{23}{144} = 0.16
\]
Outline

1. Problem formulation
2. Genetic algorithms
3. Simulated Annealing
Formally introduced in the United States in the 1970s by John Holland at University of Michigan
Loosely mimic the process of evolution of organisms, where a problem solution stands in for the organism’s genetic string
Based on:
- selection
- crossover (recombination of solutions)
- mutation
- comparison
Can be applied to any problem that has
- a solution can be expressed as a string (genome/chromosome)
- a value representing the worth of the string can be calculated.
Genetic algorithms

The algorithm repeatedly modifies a population of individual solutions

- At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation
- Over successive generations, the population "evolves" toward an optimal solution.

May solve problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, non-differentiable, stochastic, or highly nonlinear.
Three main operators in a basic genetic algorithm:

- reproduction
- crossover
- mutation

Initial population of solutions: the simplest (but probably not the best) way to create an initial population is generate it randomly.

The size of the population have to be large enough that it can support sufficient genetic variation, but not so large that calculations take an inordinate amount of time. In practice, the population size is often determined by experimentation.
Classical Algorithms vs Genetic algorithms

Classical Algorithm:

- Generates a single point at each iteration. The sequence of points approaches an optimal solution.
- Selects the next point in the sequence by a deterministic computation.

Genetic Algorithm

- Generates a population of points at each iteration. The best point in the population approaches an optimal solution.
- Selects the next population by computation which uses random number generators.
Example: person-job assignment problem

We are assigning salespeople to regions. The expected number of units sold if a salesperson is assigned to a region:

<table>
<thead>
<tr>
<th>Salesperson</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
<th>Region 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>37</td>
<td>15</td>
<td>28</td>
</tr>
<tr>
<td>B</td>
<td>25</td>
<td>24</td>
<td>18</td>
<td>29</td>
</tr>
<tr>
<td>C</td>
<td>18</td>
<td>30</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>D</td>
<td>21</td>
<td>33</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>E</td>
<td>23</td>
<td>31</td>
<td>19</td>
<td>23</td>
</tr>
</tbody>
</table>

Should be solved by linear programming!
maximize the number of units sold
We assign just 4 of the 5 salespeople (each salesperson can handle only one region)
Example: person-job assignment problem

Can genetic algorithm be applied to this problem?

- Can a solution be expressed as a string? Yes: a solution such as CDAB can represent the assignment of C to region 1, D to region 2, A to region 3 and B to region 4
- Can a value be assigned to a string to represent its value? Yes: simply add up the expect units sold for the solution (e.g., CDAB would be $18 + 33 + 15 + 29 = 95$)

At all times we will have a population consisting of numerous solution strings (genetic string of chromosomes).
Reproduction Operator

Equivalent to the survival of the fittest
The probability of survival of a solution is proportional to its solution value (its **fitness**)

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness (solution value)</th>
<th>Fitness as % of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDAB</td>
<td>95</td>
<td>95/373 = 25.5%</td>
</tr>
<tr>
<td>BADC</td>
<td>102</td>
<td>102/373 = 27.3%</td>
</tr>
<tr>
<td>BCDA</td>
<td>99</td>
<td>99/373 = 26.5%</td>
</tr>
<tr>
<td>CBAD</td>
<td>77</td>
<td>77/373 = 20.7%</td>
</tr>
<tr>
<td><em>fitness total</em></td>
<td><em>373</em></td>
<td><em>373/373 = 100.0%</em></td>
</tr>
</tbody>
</table>

How will we decide which ones survive?
After the reproduction operation, we have an intermediate population known as the mating pool that is ready to mix.
Crossover Operator

Two parent solution strings from the mating pool combine to create two new child solution strings:

1. Randomly select two parent strings from the mating pool.
2. Randomly select a crossover point in the solution string. This is the point between any two positions in the solution string.
3. Swap the ends of the two parent strings, from the crossover point to the end of the string, to create two new child strings.
Mutation Operator

Used to randomly alter the values of some of the positions in some of the strings based on a parameter that determines the level of mutation

Motivation behind mutation: to sample the solution space widely
Reproduction and crossover deal with better solutions, mutation samples the solution space and to broaden the search. It is even possible (but more inefficient) to solve problems using only the mutation operator
Overview of the Basic Genetic Algorithm Process

- Design the algorithm: choose the population size $n$ and 
  **mutation rate**; choose the **operators** and the **stopping conditions**

- Randomly generate an initial population and calculate the **fitness value** for each string. Set the incumbent solution as the solution with the best value of the fitness function in the initial population.

- Apply the **reproduction operator** to the current population to generate a mating pool of size $n$.

- Apply the **crossover operator** to the strings in the mating pool to generate a tentative new population of size $n$.

- Apply the **mutation operator** to create the final population. Calculate the fitness values.

- If the stopping conditions are met, then exit with the incumbent solution as the final solution. Otherwise go to Step 3.
Stopping Conditions

- after a respecified number of generation have been created
- when there is very little change between generations (evolutionary process has reached a plateau)
- stop when the worst solution string fitness in the population
Semi-random but reasonably good solution:

1. randomly select a salesperson and randomly assign that salesperson to a region
2. select the best unassigned salesperson-region combination and make that assignment
3. continue with step 2 until sufficient salespeople have been assigned.

or

1. randomly choose a region and assign the best salesperson for that region
2. randomly choose an unassigned region and assign the best salesperson for that region
3. continue with step 2 until there are no more regions needing a salesperson.
There are 4 and 5 white balls and 6 and 3 black balls in two boxes, respectively. You take randomly one ball from every box, and then take one of them again. What is the probability that the last ball is white?
There are 4 and 5 white balls and 6 and 3 black balls in two boxes, respectively. You take randomly one ball from every box, and then take one of them again. What is the probability that the last ball is white?

Hypotheses: $H_1 = (1 \text{ - white, } 2 \text{ - black}),$
$H_2 = (1 \text{ - white, } 2 \text{ - white}),$
$H_3 = (1 \text{ - black, } 2 \text{ - black}),$
$H_4 = (1 \text{ - black, } 2 \text{ - white})$

$P(H_1) = \frac{4}{4+6} \cdot \frac{3}{5+3} = \frac{12}{80}$
$P(H_1) = \frac{4}{4+6} \cdot \frac{5}{5+3} = \frac{12}{80}$
$P(H_1) = \frac{6}{4+6} \cdot \frac{3}{5+3} = \frac{118}{80}$
$P(H_1) = \frac{6}{4+6} \cdot \frac{5}{5+3} = \frac{30}{80}$

$P(A|H_1) = P(A|H_4) = \frac{1}{2}, P(A|H_2) = 1, P(A|H_3) = 1$

$P(A) = \frac{12}{80} \cdot \frac{1}{2} + \frac{12}{80} \cdot 0 + \frac{118}{80} + \frac{30}{80} \cdot \frac{1}{2} = \frac{41}{80}$
Outline

1. Problem formulation
2. Genetic algorithms
3. Simulated Annealing
Simulated Annealing

Another popular heuristic for both \textit{discrete} and continuous, unconstrained and bound-constrained optimization problems (described in 1983-1985)
Mimics the physical process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy

- At each iteration a new point is randomly generated
- The distance of the new point from the current point, or the extent of the search, is based on a probability distribution with a scale proportional to the temperature
- The algorithm accepts all new points that lower the objective, but also, with a certain probability, points that raise the objective

By accepting points that raise the objective, the algorithm avoids being trapped in local minima in early iterations and is able to explore globally for better solutions.
Outline of this simple algorithm

minimization of a cost function

1. Start-up. Find an initial solution $S$ (possibly randomly), initial (high) temperature $T > 0$, the rate of cooling parameter $r$.

2. Choose a random neighbor of $S$ and call it $S'$.

3. Calculate the difference in costs: $\Delta = cost(S') - cost(S)$.

4. Decide whether to accept the new solution or not: if $\Delta \leq 0$ ( $S'$ is better) then $S = S'$, else set $S = S'$ with probability $e^{-\Delta/T}$.

5. If the stopping conditions are met, then exit with $S$ as the final solution, else reduce the temperature by setting $T = rT$, and go to Step 2.

Stopping condition is when $S$ is 'frozen' (has not changed value for several iterations).
Summary

- local optimum
- mixed discrete/continuous problems
- various data representation
- stochastic
- implementation is still an art
- no gradient or fancy math/objective function designing may be difficult
- computationally expensive
- easy parallelized