Genetic Algorithms

Kang Zheng
Karl Schober

Genetic algorithm

- What is Genetic algorithm?

- A genetic algorithm (or GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- (GA)s are categorized as global search heuristics.
- (GA)s are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

What is Genetic algorithm?

- Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions.
- Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

What is Genetic algorithm?

- The evolution usually starts from a population of randomly generated individuals and happens in generations.
- In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified (recombined and possibly mutated) to form a new population.

What is Genetic algorithm?

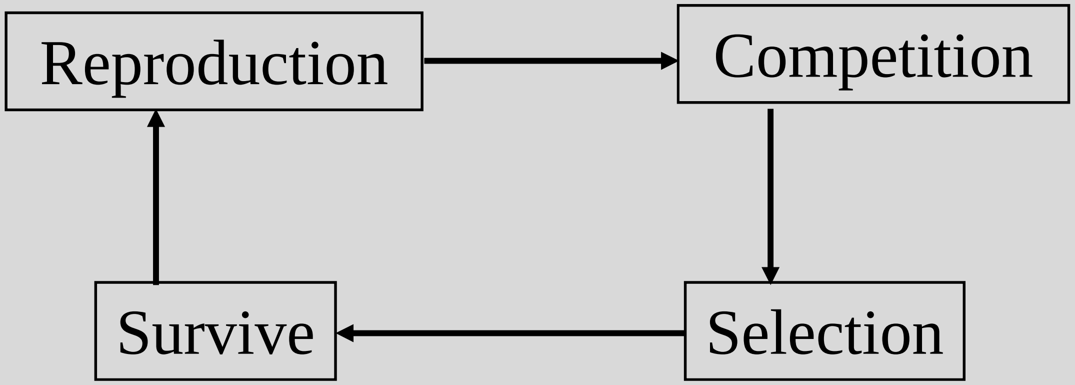
- The new population is then used in the next iteration of the algorithm.
- Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.
- If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Key terms

- **Individual** - Any possible solution
- **Population** - Group of all *individuals*
- **Fitness** - Target function that we are optimizing ((each individual has a fitness)
- **Trait** - Possible aspect (*features)* of an *individual*
- **Genome** - Collection of all *chromosomes* for an *individual*

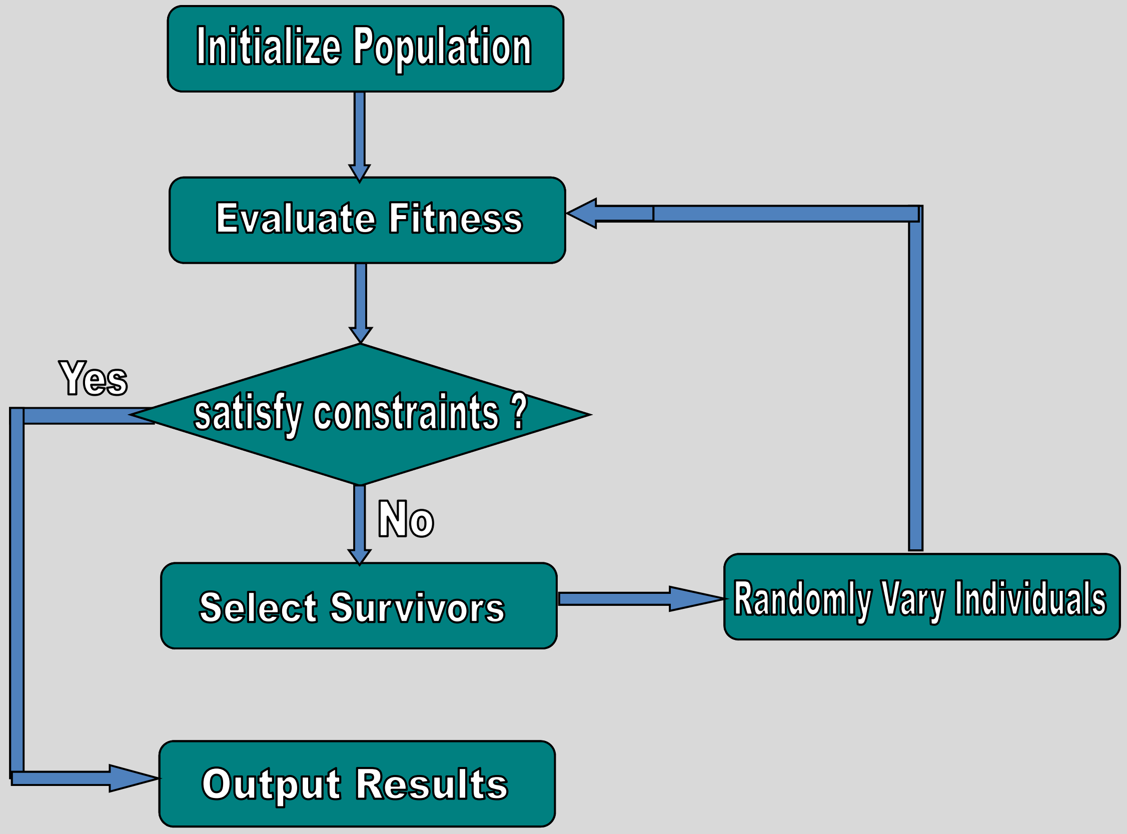
Genetic algorithm

- Based on Darwinian Paradigm



- Intrinsically a robust search and optimization mechanism

Genetic algorithm



Example: the MAXONE problem

- Suppose we want to maximize the number of ones in a string of / binary digits

Is it a trivial problem?

- It may seem so because we know the answer in advance
- However, we can think of it as maximizing the number of correct answers, each encoded by 1, to / yes/no difficult questions.

Example (cont)

- An individual is encoded (naturally) as a string of l binary digits
- The fitness f of a candidate solution to the MAXONE problem is the number of ones in its genetic code
- We start with a population of n random strings. Suppose that $l = 10$ and $n = 6$

Example (initialization)

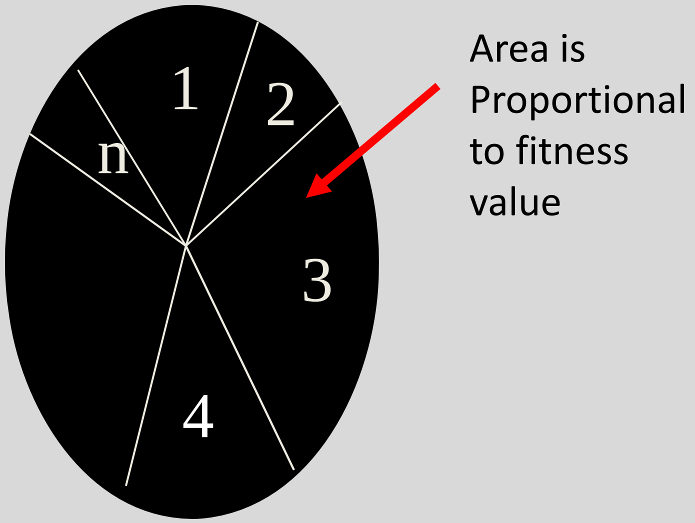
- We toss a fair coin 60 times and get the following initial population:
- $s_1 = 1111010101 \quad f(s_1) = 7$
- $s_2 = 0111000101 \quad f(s_2) = 5$
- $s_3 = 1110110101 \quad f(s_3) = 7$
- $s_4 = 0100010011 \quad f(s_4) = 4$
- $s_5 = 1110111101 \quad f(s_5) = 8$
- $s_6 = 0100110000 \quad f(s_6) = 3$

Example (selection1)

- Next we apply fitness proportionate selection with the roulette wheel method:

Individual i will have a $\frac{f(i)}{\sum_i f(i)}$ probability to be chosen

We repeat the extraction as many times as the number of individuals we need to have the same parent population size (6 in our case)



Example (selection2)

- Suppose that, after performing selection, we get the following population:
- s_1` = 1111010101 (s_1)
- s_2` = 1110110101 (s_3)
- s_3` = 1110111101 (s_5)
- s_4` = 0111000101 (s_2)
- s_5` = 0100010011 (s_4)
- s_6` = 1110111101 (s_5)

Example (crossover1)

- Next we mate strings for crossover. For each couple we decide according to crossover probability (for instance 0.6) whether to actually perform crossover or not
- Suppose that we decide to actually perform crossover only for couples (s_1\`, $s_2$\`) and (s_5\`, $s_6$\`). For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second

Example (crossover2)

Before crossover:

$s_1` = 1111010101$ $s_5` = 0100010011$

$s_2` = 1110110101$ $s_6` = 1110111101$

After crossover:

$s_1`` = 1110110101$ $s_5`` = 0100011101$

$s_2`` = 1111010101$ $s_6`` = 1110110011$

Example (mutation1)

- The final step is to apply random mutation: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)
- Before applying mutation:
- $s_1``$ = 1110110101
- $s_2``$ = 1111010101
- $s_3``$ = 1110111101
- $s_4``$ = 0111000101
- $s_5``$ = 0100011101
- $s_6``$ = 1110110011

Example (mutation2)

- After applying mutation:
- s_1``` = 1110100101 $f(s_1$``` $) = 6$
- s_2``` = 1111110100 $f(s_2$``` $) = 7$
- s_3``` = 1110101111 $f(s_3$``` $) = 8$
- s_4``` = 0111000101 $f(s_4$``` $) = 5$
- s_5``` = 0100011101 $f(s_5$``` $) = 5$
- s_6``` = 1110110001 $f(s_6$``` $) = 6$

Example (end)

- In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%
- At this point, we go through the same process all over again, until a stopping criterion is met

Components of a GA

A problem definition as input, and

- Encoding principles (gene, chromosome)
- Initialization procedure (creation)
- Selection of parents (reproduction)
- Genetic operators (mutation, recombination)
- Evaluation function (environment)
- Termination condition

The Bin Packing Problem

- Set of n items with weights
- Bins have a fixed capacity
- The size of a solution is the number of bins with items inside
- Try to minimize the solution size

Genetic Encoding

- Bin-based representation
  - Fixed length, each gene represents one item and where it is packed
- Object-based representation
  - Permutations of items
- Group-based representation
  - Each gene represents an occupied bin and its group of items

Fitness Function

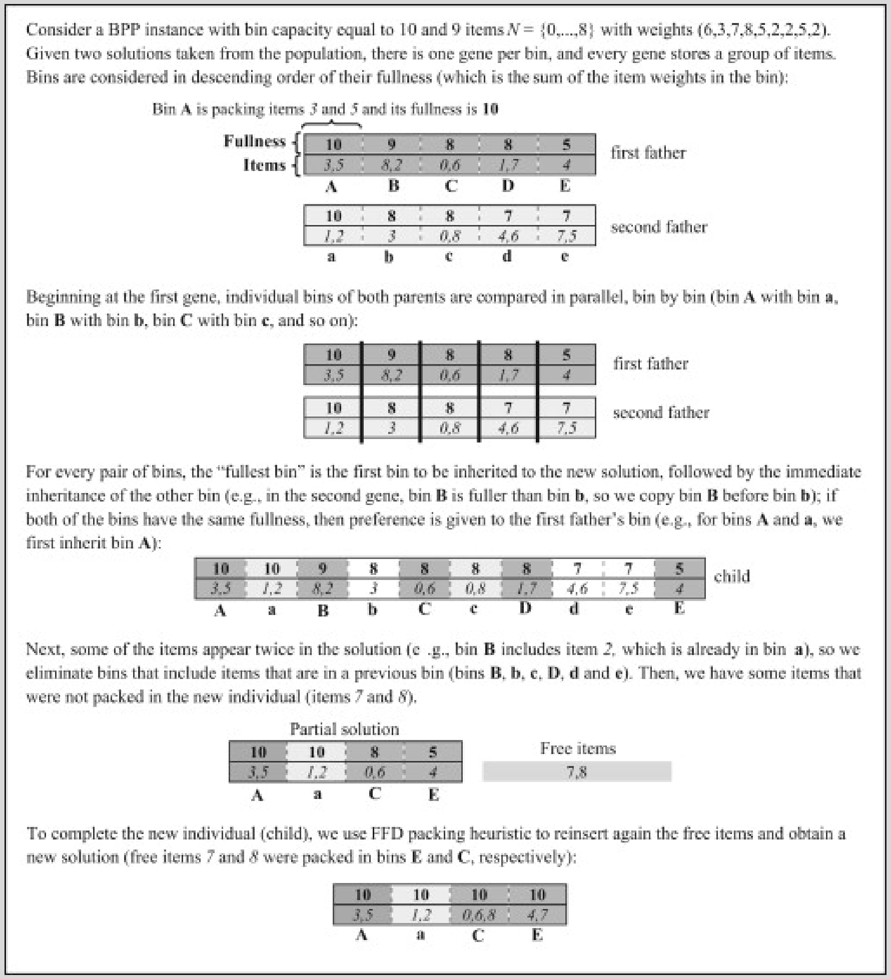
- $F_{BPP} = \frac{\sum_{i=1}^{m}(S_i/c)^2}{m}$
- S_i - sum of item weights
- c – bin capacity
- A combination of some nearly full bins and some nearly empty bins is better than equally filled bins

Initial Population

- Items with size larger than fifty percent of the cost are placed into separate bins which are randomly ordered
- Each remaining item is then placed in the first bin which has enough capacity, or a new empty bin if none are available

Crossover Operator

- Want to propagate features which contribute the most to fitness
- Three approaches considered
  - Segment crossover
  - Gene crossover
  - Ordered gene crossover



Consider a BPP instance with bin capacity equal to 10 and 9 items $N = \{0,...,8\}$ with weights (6,3,7,8,5,2,2,5,2). Given two solutions taken from the population, there is one gene per bin, and every gene stores a group of items. Bins are considered in descending order of their fullness (which is the sum of the item weights in the bin):

Bin **A** is packing items *3* and *5* and its fullness is **10**

Fullness	10	9	8	8	5	first father
Items	*3,5*	*8,2*	*0,6*	*1,7*	*4*	
	A	**B**	**C**	**D**	**E**	

10	8	8	7	7	second father
1,2	*3*	*0,8*	*4,6*	*7,5*	
a	**b**	**c**	**d**	**e**	

Beginning at the first gene, individual bins of both parents are compared in parallel, bin by bin (bin **A** with bin **a**, bin **B** with bin **b**, bin **C** with bin **c**, and so on):

10	9	8	8	5	first father
3,5	*8,2*	*0,6*	*1,7*	*4*	
10	8	8	7	7	second father
1,2	*3*	*0,8*	*4,6*	*7,5*	

For every pair of bins, the "fullest bin" is the first bin to be inherited to the new solution, followed by the immediate inheritance of the other bin (e.g., in the second gene, bin **B** is fuller than bin **b**, so we copy bin **B** before bin **b**); if both of the bins have the same fullness, then preference is given to the first father's bin (e.g., for bins **A** and **a**, we first inherit bin **A**):

10	10	9	8	8	8	8	7	7	5	child
3,5	*1,2*	*8,2*	*3*	*0,6*	*0,8*	*1,7*	*4,6*	*7,5*	*4*	
A	**a**	**B**	**b**	**C**	**c**	**D**	**d**	**e**	**E**	

Next, some of the items appear twice in the solution (e .g., bin **B** includes item *2,* which is already in bin **a**), so we eliminate bins that include items that are in a previous bin (bins **B**, **b**, **c**, **D**, **d** and **e**). Then, we have some items that were not packed in the new individual (items *7* and *8*).

Partial solution

10	10	8	5
3,5	*1,2*	*0,6*	*4*
A	**a**	**C**	**E**

Free items

7,8

To complete the new individual (child), we use FFD packing heuristic to reinsert again the free items and obtain a new solution (free items *7* and *8* were packed in bins **E** and **C**, respectively):

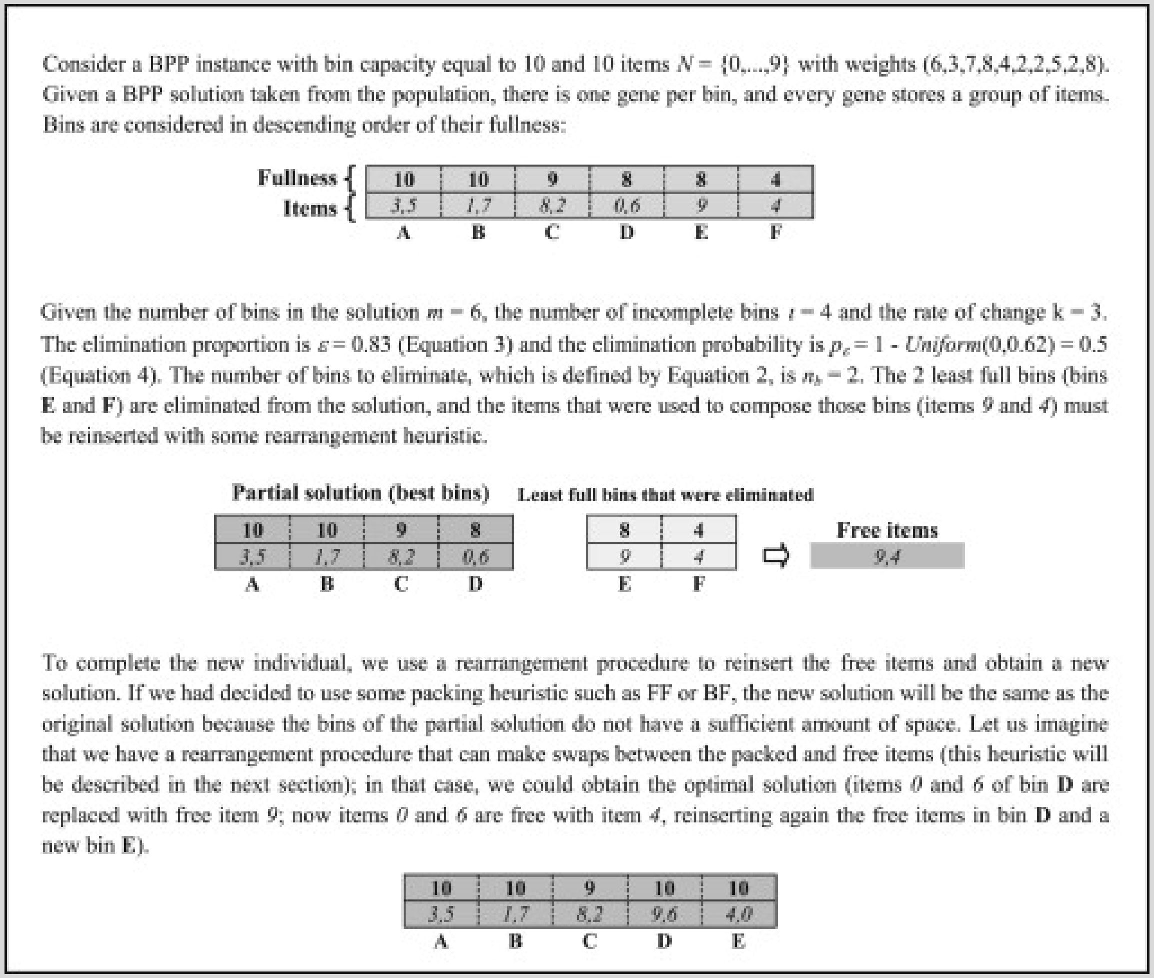
10	10	10	10
3,5	*1,2*	*0,6,8*	*4,7*
A	**a**	**C**	**E**

Fig. 3 An example of the gene-level crossover operator that preserves the fullest-bin pattern.

http://dx.doi.org/10.1016/j.cor.2014.10.010

Mutation Operator

- Introduce random changes in the population at a low rate to increase fitness
- Remove some bins and replace their items
- Make the probability of elimination for an individual bin higher for a less filled bin
- Eliminate more bins in less fit solutions



Consider a BPP instance with bin capacity equal to 10 and 10 items $N = \{0,\ldots,9\}$ with weights (6,3,7,8,4,2,2,5,2,8). Given a BPP solution taken from the population, there is one gene per bin, and every gene stores a group of items. Bins are considered in descending order of their fullness:

Fullness	**10**	**10**	**9**	**8**	**8**	**4**
Items	*3,5*	*1,7*	*8,2*	*0,6*	*9*	*4*
	A	**B**	**C**	**D**	**E**	**F**

Given the number of bins in the solution $m = 6$, the number of incomplete bins $\iota = 4$ and the rate of change k = 3. The elimination proportion is $\varepsilon = 0.83$ (Equation 3) and the elimination probability is $p_\varepsilon = 1$ - *Uniform*(0,0.62) = 0.5 (Equation 4). The number of bins to eliminate, which is defined by Equation 2, is $n_b = 2$. The 2 least full bins (bins **E** and **F**) are eliminated from the solution, and the items that were used to compose those bins (items *9* and *4*) must be reinserted with some rearrangement heuristic.

Partial solution (best bins)

10	**10**	**9**	**8**
3,5	*1,7*	*8,2*	*0,6*
A	**B**	**C**	**D**

Least full bins that were eliminated

8	**4**
9	*4*
E	**F**

$\Rightarrow$ **Free items**

9,4

To complete the new individual, we use a rearrangement procedure to reinsert the free items and obtain a new solution. If we had decided to use some packing heuristic such as FF or BF, the new solution will be the same as the original solution because the bins of the partial solution do not have a sufficient amount of space. Let us imagine that we have a rearrangement procedure that can make swaps between the packed and free items (this heuristic will be described in the next section); in that case, we could obtain the optimal solution (items *0* and *6* of bin **D** are replaced with free item *9*; now items *0* and *6* are free with item *4*, reinserting again the free items in bin **D** and a new bin **E**).

10	**10**	**9**	**10**	**10**
3,5	*1,7*	*8,2*	*9,6*	*4,0*
A	**B**	**C**	**D**	**E**

Fig. 4 An example of the adaptive mutation operator that preserves the fullest-bin pattern.

http://dx.doi.org/10.1016/j.cor.2014.10.010

Rearrangement Heuristics

- Need to swap packed and free items during replacement in the mutation operator
- Swap free items into a bin if it does not decrease fullness
- Consider single swaps and pairs of free items
- Use the first fit packing to replace anything left over

Reproduction Technique

- Selection
  - Elite population of individuals is kept to ensure selective pressure
  - Pair elite individuals with ones chosen at random from the population
    - Promotes survival of good genes as well as diversity
  - Mutate "young" solutions in the elite population

Reproduction Technique

- Replacement
  - Age-based replacement
    - Replace the oldest solutions
  - Fitness-based replacement
    - Replace the worst solutions
  - Random-based replacement
    - Replace random solutions
  - Age-based and random-based replacement can lose best solutions, fitness-based can lead to premature convergence of solutions.

Results

- The authors conclude that their algorithm produces good results across several variations of the problem
- There is more work to be done for tuning of the many parameters of their algorithm

References

- Marcela Quiroz-Castellanos, Laura Cruz-Reyes, Jose Torres-Jimenez, Claudia Gómez S., Héctor J. Fraire Huacuja, Adriana C.F. Alvim, A grouping genetic algorithm with controlled gene transmission for the bin packing problem, Computers & Operations Research, Volume 55, March 2015, Pages 52-64, ISSN 0305-0548, http://dx.doi.org/10.1016/j.cor.2014.10.010.