

Genetic Algorithms

Chapter 3

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Genetic Algorithms

GA 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
 - good heuristic for combinatorial problems
- Special Features:
 - Traditionally emphasizes combining information from good parents (crossover)
 - many variants, e.g., reproduction models, operators

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Genetic Algorithms

Genetic algorithms

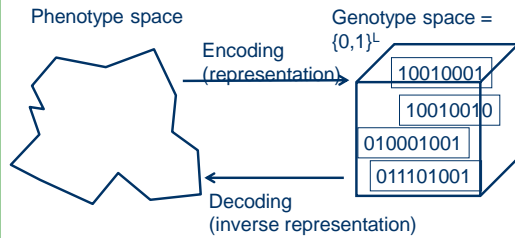
- Holland's original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
 - Representations
 - Mutations
 - Crossovers
 - Selection mechanisms

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Genetic Algorithms

SGA technical summary tableau

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

Representation

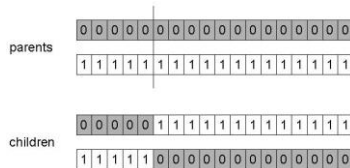


SGA reproduction cycle

1. Select parents for the mating pool
(size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring

SGA operators: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



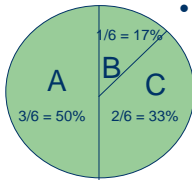
SGA operators: mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
 - Typically between $1/\text{pop_size}$ and $1/\text{chromosome_length}$



SGA operators: Selection

- Main idea: better individuals get higher chance
 - Chances proportional to fitness
 - Implementation: roulette wheel technique
 - Assign to each individual a part of the roulette wheel
 - Spin the wheel n times to select n individuals



fitness(A) = 3
fitness(B) = 1
fitness(C) = 2

An example after Goldberg '89 (1)

- Simple problem: $\max x^2$ over $\{0, 1, \dots, 31\}$
- GA approach:
 - Representation: binary code, e.g. $01101 \leftrightarrow 13$
 - Population size: 4
 - 1-point crossover, bitwise mutation
 - Roulette wheel selection
 - Random initialisation
- We show one generational cycle done by hand

x^2 example: selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

x^2 example: crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 1	4	0 1 1 0 0	12	144
2	1 1 0 0 0	4	1 1 0 0 1	25	625
2	1 1 0 0 0	2	1 1 0 1 1	27	729
4	1 0 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X² example: mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple GA

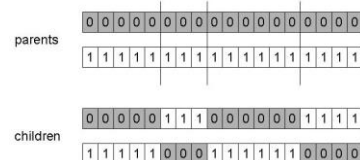
- Has been subject of many (early) studies
 - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
 - Representation is too restrictive
 - Mutation & crossovers only applicable for bit-string & integer representations
 - Selection mechanism sensitive for converging populations with close fitness values
 - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

Alternative Crossover Operators

- Performance with 1 Point Crossover depends on the order that variables occur in the representation
 - more likely to keep together genes that are near each other
 - Can never keep together genes from opposite ends of string
 - This is known as *Positional Bias*
 - Can be exploited if we know about the structure of our problem, but this is not usually the case

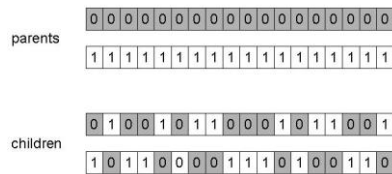
n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)



Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



Integer representations

- Some problems naturally have integer variables, e.g. image processing parameters
- Others take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
 - "creep" i.e. more likely to move to similar value
 - Random choice (esp. categorical variables)
 - For ordinal problems, it is hard to know correct range for creep, so often use two mutation operators in tandem

Floating point mutations 1

General scheme of floating point mutations

$$\bar{x} = \langle x_1, \dots, x_i \rangle \rightarrow \bar{x}' = \langle x'_1, \dots, x'_i \rangle$$

$$x_i, x'_i \in [LB_i, UB_i]$$

- Uniform mutation:
 - x'_i drawn randomly (uniform) from $[LB_i, UB_i]$
- Analogous to bit-flipping (binary) or random resetting (integers)

Floating point mutations 2

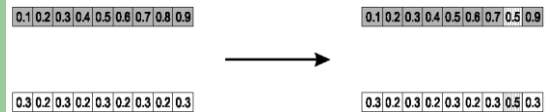
- Non-uniform mutations:
 - Many methods proposed, such as time-varying range of change etc.
 - Most schemes are probabilistic but usually only make a small change to value
 - Most common method is to add random deviate to each variable separately, taken from $N(0, \sigma)$ Gaussian distribution and then curtail to range
 - Standard deviation σ controls amount of change (2/3 of deviations will lie in range $(-\sigma$ to $+\sigma)$)

Crossover operators for real valued GAs

- Discrete:
 - each allele value in offspring z comes from one of its parents (x, y) with equal probability: $z_i = x_i$ or y_i
 - Could use n-point or uniform
- Intermediate
 - exploits idea of creating children “between” parents (hence a.k.a. *arithmetic* recombination)
 - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
 - The parameter α can be:
 - constant: uniform arithmetical crossover
 - variable (e.g. depend on the age of the population)
 - picked at random every time

Single arithmetic crossover

- Parents: $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$
- Pick a single gene (k) at random,
- child₁ is: $\langle x_1, \dots, x_k, \alpha \cdot y_k + (1 - \alpha) \cdot x_k, \dots, x_n \rangle$
- reverse for other child. e.g. with $\alpha = 0.5$



Simple arithmetic crossover

- Parents: $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$
- Pick random gene (k) after this point mix values
- child₁ is:

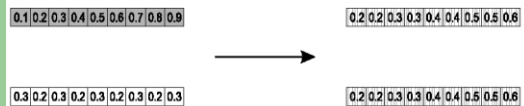
$$\langle x_1, \dots, x_k, \alpha \cdot y_{k+1} + (1 - \alpha) \cdot x_{k+1}, \dots, \alpha \cdot y_n + (1 - \alpha) \cdot x_n \rangle$$
- reverse for other child. e.g. with $\alpha = 0.5$



Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$
- child₁ is:

$$a \cdot \bar{x} + (1 - a) \cdot \bar{y}$$
- reverse for other child. e.g. with $\alpha = 0.5$

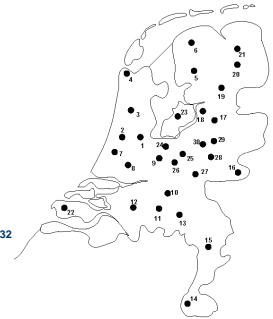


Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
 - Example: sort algorithm: important thing is which elements occur before others (order)
 - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a permutation:
 - if there are n variables then the representation is as a list of n integers, each of which occurs exactly once

Permutation representation: TSP example

- Problem:
 - Given n cities
 - Find a complete tour with minimal length
- Encoding:
 - Label the cities $1, 2, \dots, n$
 - One complete tour is one permutation (e.g. for $n=4$ [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:
 - for 30 cities there are $30! \approx 10^{32}$ possible tours



Mutation operators for permutations

- Normal mutation operators lead to inadmissible solutions
 - e.g. bit-wise mutation : let gene j have value j
 - changing to some other value k would mean that k occurred twice and j no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

Insert Mutation for permutations

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information

1 2 3 4 5 6 7 8 9 → 1 2 5 3 4 6 7 8 9

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Genetic Algorithms

Swap mutation for permutations

- Pick two alleles at random and swap their positions
- Preserves most of adjacency information (4 links broken), disrupts order more

1 2 3 4 5 6 7 8 9 → 1 5 3 4 2 6 7 8 9

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Genetic Algorithms

Inversion mutation for permutations

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information

1 2 3 4 5 6 7 8 9 → 1 5 4 3 2 6 7 8 9

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Genetic Algorithms

Scramble mutation for permutations

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions

1 2 3 4 5 6 7 8 9 → 1 3 5 4 2 6 7 8 9

(note subset does not have to be contiguous)

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Genetic Algorithms

Crossover operators for permutations

- “Normal” crossover operators will often lead to inadmissible solutions



- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Partially Mapped Crossover (PMX)

Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these i look in the offspring to see what element j has been copied in its place from P1
4. Place i into the position occupied j in P2, since we know that we will not be putting j there (as is already in offspring)
5. If the place occupied by j in P2 has already been filled in the offspring k , put i in the position occupied by k in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

PMX example

- Step 1

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

→

			4	5	6	7	
--	--	--	---	---	---	---	--
- Step 2

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

→

	2	4	5	6	7		8
--	---	---	---	---	---	--	---
- Step 3

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

→

9	3	2	4	5	6	7	1	8
---	---	---	---	---	---	---	---	---

Population Models

- SGA uses a Generational model:
 - each individual survives for exactly one generation
 - the entire set of parents is replaced by the offspring
- At the other end of the scale are Steady-State models:
 - one offspring is generated per generation,
 - one member of population replaced,
- Generation Gap
 - the proportion of the population replaced
 - 1.0 for GGA, $1/\text{pop_size}$ for SSGA

Fitness Based Competition

- Selection can occur in two places:
 - Selection from current generation to take part in mating (parent selection)
 - Selection from parents + offspring to go into next generation (survivor selection)
- Selection operators work on whole individual
 - i.e. they are representation-independent
- Distinction between selection
 - operators: define selection probabilities
 - algorithms: define how probabilities are implemented

Implementation example: SGA

- Expected number of copies of an individual i

$$E(n_i) = \mu \cdot f(i) / \langle f \rangle$$

(μ = pop.size, $f(i)$ = fitness of i , $\langle f \rangle$ avg. fitness in pop.)

- Roulette wheel algorithm:

- Given a probability distribution, spin a 1-armed wheel n times to make n selections
- No guarantees on actual value of n_i

- Baker's SUS algorithm:

- n evenly spaced arms on wheel and spin once
- Guarantees $\text{floor}(E(n_i)) \leq n_i \leq \text{ceil}(E(n_i))$

Fitness-Proportionate Selection

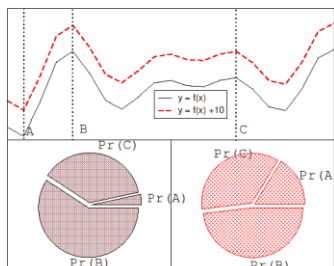
- Problems include

- One highly fit member can rapidly take over if rest of population is much less fit: Premature Convergence
- At end of runs when fitnesses are similar, lose selection pressure
- Highly susceptible to function transposition

- Scaling can fix last two problems

- Windowing: $f(i) = f(i) - \beta^t$
 - where β is worst fitness in this (last n) generations
- Sigma Scaling: $f(i) = \max(f(i) - (\langle f \rangle - c \cdot \sigma_f), 0.0)$
 - c is a constant, usually 2.0, $\langle f \rangle$ = mean(f), σ_f stand dev.

Function transposition for FPS



Rank – Based Selection

- Attempt to remove problems of FPS by basing selection probabilities on *relative* rather than *absolute* fitness
- Rank population according to fitness and then base selection probabilities on rank where fittest has rank μ and worst rank 1 (expected # of offspring)
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Linear Ranking

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- Parameterised by factor s : $1.0 < s \leq 2.0$
 - measures advantage of best individual
 - in GGA this is the number of children allotted to it
- Simple 3 member example

	Fitness	Rank	P_{selFP}	$P_{selLR} (s=2)$	$P_{selLR} (s=1.5)$
A	1	1	0.1	0	0.167
B	5	2	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

Tournament Selection

- All methods above rely on global population statistics
 - Could be a bottleneck esp. on parallel machines
 - Relies on presence of external fitness function which might not exist: e.g. evolving game players
- Informal Procedure:
 - Pick k members at random then select the best of these
 - Repeat to select more individuals

Tournament Selection 2

- Probability of selecting i will depend on:
 - Rank of i
 - Size of sample k
 - higher k increases selection pressure
 - Whether fittest contestant always wins (deterministic) or this happens with probability p
 - Whether contestants are picked with replacement
 - Picking without replacement increases selection pressure
 - Especially in deterministic tournaments
- For $k=2$, time for fittest individual to take over population is the same as linear ranking with $s=2 \cdot p$

Survivor Selection

- Most of methods above used for parent selection
- Survivor selection can be divided into two approaches:
 - Age-Based Selection
 - e.g. SGA
 - In SSGA can implement as "delete-random" (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
 - Fitness-Based Selection
 - Using one of the methods above

Two Special Cases

- Elitism
 - Widely used in both population models (GGA, SSGA)
 - Always keep at least one copy of the fittest solution so far
- GENITOR: a.k.a. “delete-worst”
 - From Whitley’s original Steady-State algorithm (he also used linear ranking for parent selection)
 - Rapid takeover : use with large populations or “no duplicates” policy

Example application of order based GAs: JSSP

Precedence constrained job shop scheduling problem

- J is a set of jobs.
- O is a set of operations
- M is a set of machines
- $Able \subseteq O \times M$ defines which machines can perform which operations
- $Pre \subseteq O \times O$ defines which operation should precede which
- $Dur : \subseteq O \times M \rightarrow \mathbb{R}$ defines the duration of $o \in O$ on $m \in M$

The goal is now to find a schedule that is:

- Complete: all jobs are scheduled
- Correct: all conditions defined by $Able$ and Pre are satisfied
- Optimal: the total duration of the schedule is minimal

Precedence constrained job shop scheduling GA

- Representation: individuals are permutations of operations
- Permutations are decoded to schedules by a decoding procedure
 - take the first (next) operation from the individual
 - look up its machine (here we assume there is only one)
 - assign the earliest possible starting time on this machine, subject to
 - machine occupation
 - precedence relations holding for this operation in the schedule created so far
- fitness of a permutation is the duration of the corresponding schedule (to be minimized)
- use any suitable mutation and crossover
- use roulette wheel parent selection on inverse fitness
- Generational GA model for survivor selection
- use random initialisation

JSSP example: operator comparison

