CIS 189

Lecture 13: Genetic Algorithms

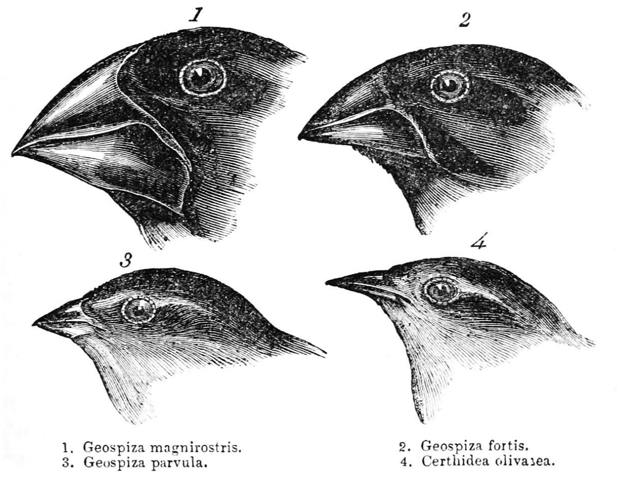
Rohan Menezes rohanmenezes@alumni.upenn.edu

Final Projects

- Check-in extended to Thursday at 4 pm
- Presentations next class
- Submissions due the same day at midnight

Genetic Algorithms

- Class of heuristic search algorithms inspired by Darwin's theory of evolution
- Survival of the fittest
- "Evolve" population of candidate solutions in a search space

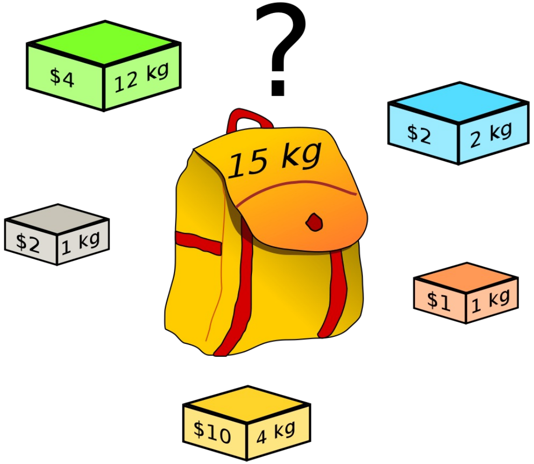


Basic Elements

- Encoding of a candidate solution as a string of characters (**genes**) from a finite alphabet
- A string of genes defines an **individual**
- A **population** is a set of N individuals
- A **fitness function** maps an individual to a fitness score, indicating the quality of that candidate solution

The 0-1 Knapsack Problem

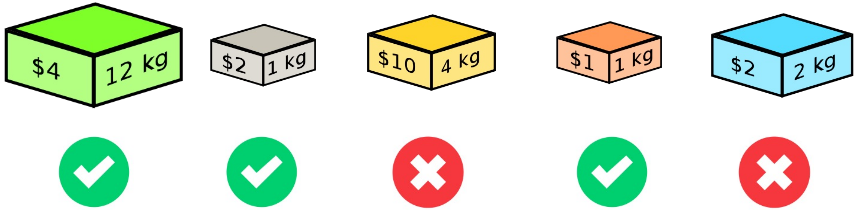
- Given n items with values $v_1, \dots, v_n$ and weights $w_1, \dots w_n$, select maximum-value subset to fit into a knapsack with capacity W.



Knapsack Genes

- Alphabet: $\{0,1\}$ (binary)
- Example candidate solution encoding:

1 1 0 1 0

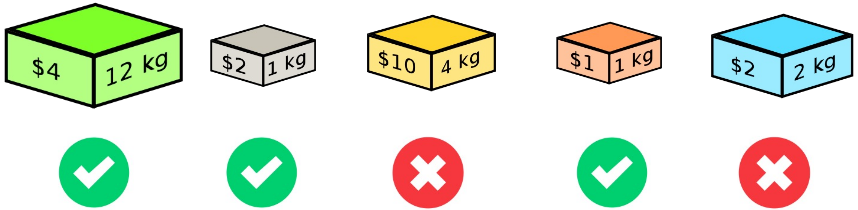


(Totals: $7 and 14kg)

Knapsack Fitness

- Fitness function: $\sum_i b_i v_i$ if $\sum_i b_i w_i \leq W$ else 0
- "The total knapsack value, or 0 if capacity is exceeded"

1 1 0 1 0



(Totals: $7 and 14kg; **fitness: 7**)

Knapsack Population

- Initial random population (generation 0):

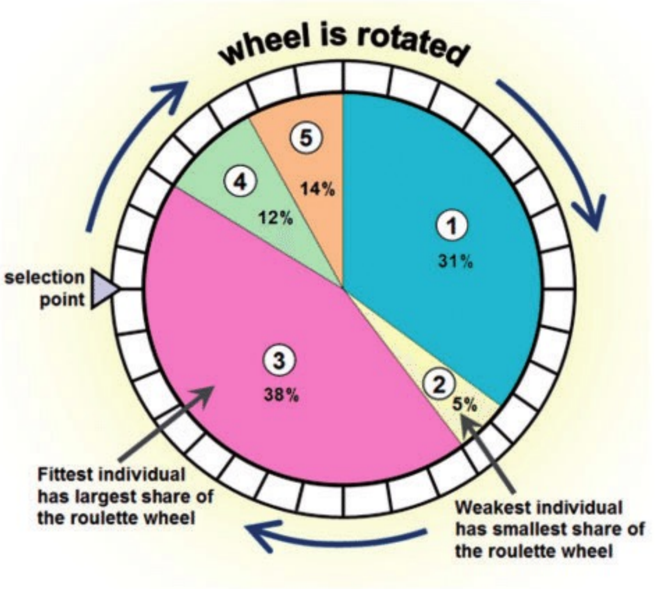
Genome	Weight	Value	Fitness
1 1 1 0 0	17kg	$16	0
0 0 1 0 0	4kg	$10	10
0 1 0 0 1	3kg	$4	4
1 1 0 0 1	15kg	$8	8

Basic Steps

- Start with an initial population
- Randomly select individuals to survive and reproduce, based on fitness
- Combine and/or mutate selected individuals to generate a new population (the next **generation**)
- Eventually, return the best found individual

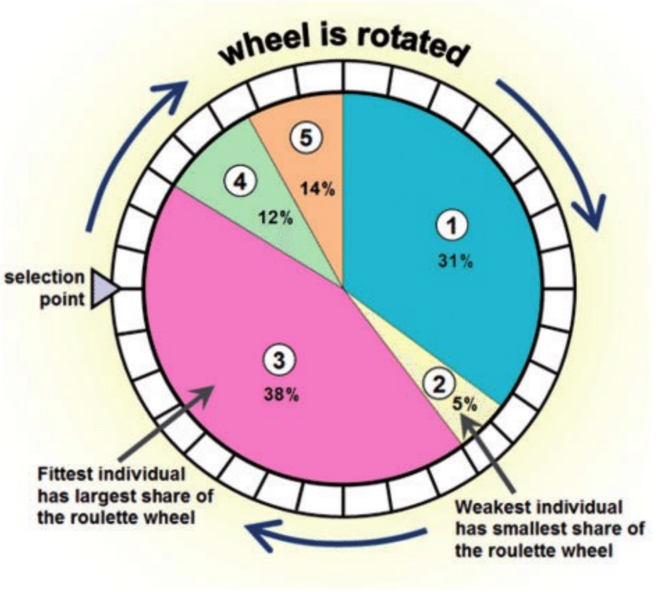
Fitness Proportionate Selection

- “Roulette wheel selection”
- Spin wheel N times, select with replacement
  - Duplicates allowed



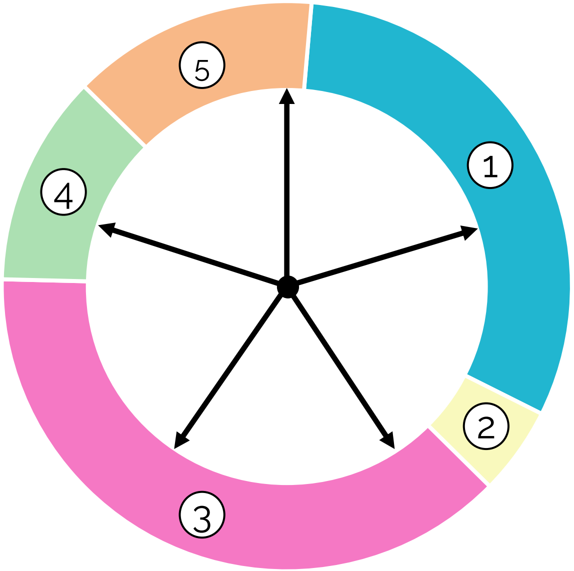
Fitness Proportionate Selection

- "Expected value" of an individual (expected # of selections)
  - $= N \dfrac{\text{fitness}}{\text{total fitness}}$
  - $= \dfrac{\text{fitness}}{\text{avg. fitness}}$
- Notation: $ExpVal(i,t) = \dfrac{f_t(i)}{\bar{f}_t}$



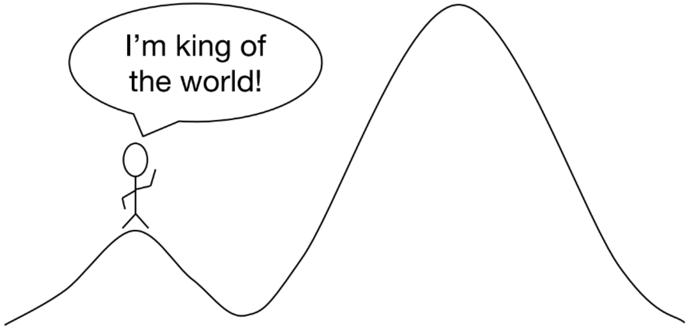
Stochastic Universal Sampling (SUS)

- Make all selections in one spin of wheel with N evenly-spaced pointers
- Reduce variance in selection
- Same expected values
- Every above-average member is guaranteed to be selected at least once

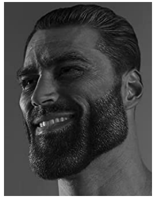


Problem: Premature Convergence

- Collapse of population diversity early on
- Caused by favoring exploitation over exploration too heavily
  - GA becomes simple “hill-climbing” algorithm

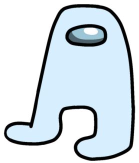


Sigma Scaling

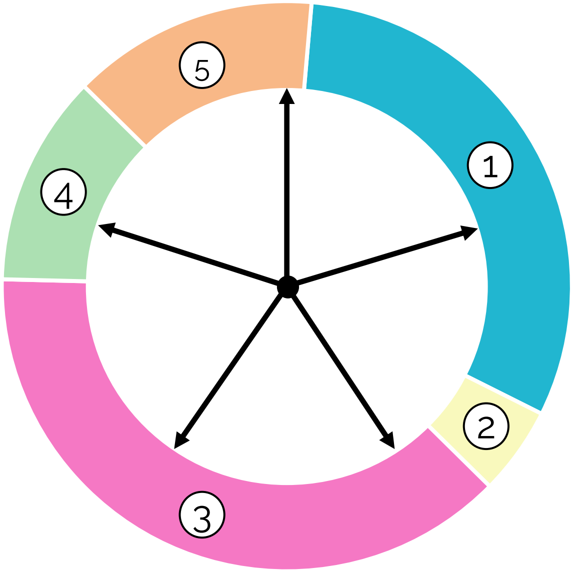


- Hold rate of exploitation relatively constant
  - Rather than depending on fitness variance
- $ExpVal(i,t) = \begin{cases} \max\left(1 + \frac{f_t(i) - \bar{f}_t}{2\sigma(t)}, 0.1\right) \text{ if } \sigma(t) > 0 \\ 1 \text{ otherwise} \end{cases}$
- Minimum expected value arbitrarily set to 0.1
  - Give very low fitness individuals a chance

Stochastic Universal Sampling (SUS)



- How to implement a desired expected value distribution?
  - $ExpVal(i, t) = N * (\text{wheel }\%)$
  - $\text{wheel }\% = \frac{ExpVal(i,t)}{N}$



Rank Selection

- Select individuals based on fitness rank (not value)
- Eliminates need for fitness scaling
  - Absolute differences in fitnesses are ignored
- Use linearly (or exponentially) decaying expected values based on rank
- $\sum_i ExpVal(i, t) = N$

Tournament Selection

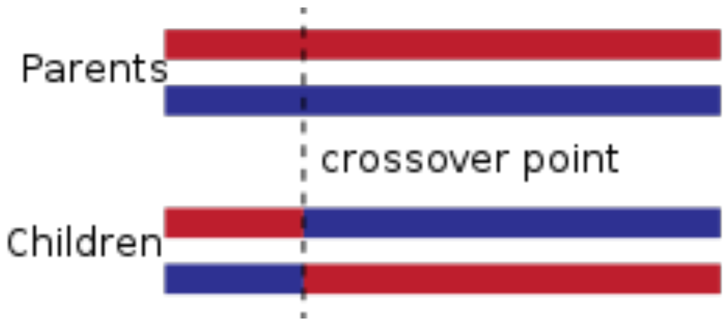
- Choose 2 individuals at random
- Select the more fit individual with probability k, and the less fit individual otherwise
  - k is a hyperparameter, e.g. 0.75
- Continue selecting (with replacement) N times

Genetic Operators

- Once we’ve selected individuals for survival/reproduction, how do we create the next generation?

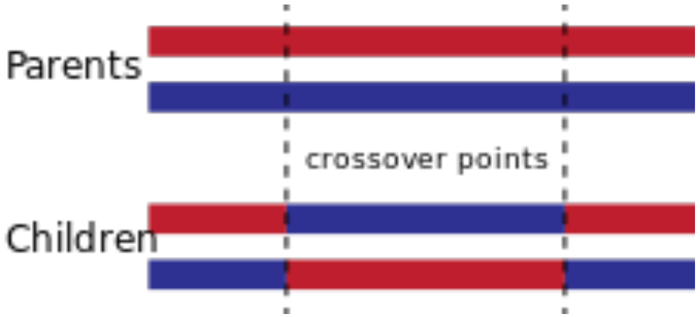
Crossover

- Combining the attributes of 2 (randomly paired) parents



Multi-point Crossover

- Combining the attributes of 2 (randomly paired) parents



Uniform Crossover

- Combining the attributes of 2 (randomly paired) parents
- For each child, choose each bit from either parent with equal probability

Mutation

- For each attribute (gene), with some small mutation probability, make a random modification
- Helps maintain genetic diversity, exploration
- For alphabets with notion of ordering or distance, magnitude of mutation is important
    - Controlled by a hyperparameter (like “step size”)

Elitism

- Force the GA to retain some number of the best individuals at each generation
- Prevent random exclusion from selection, as well as destruction from crossover or mutation

Termination

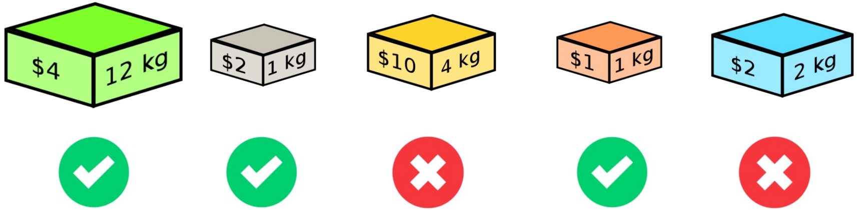
Some options:

- X number of generations completed - typically 100s
- Threshold on σ_t (standard deviation of fitness scores)
- Threshold on best fitness improvement

Solving Knapsack with a GA

- Fitness function: $\sum_i b_i v_i$ if $\sum_i b_i w_i \leq W$ else 0
- "The total knapsack value, or 0 if capacity is exceeded"

1 1 0 1 0



(Totals: $7 and 14kg; **fitness: 7**)

Solving Knapsack with a GA

- Initial random population (generation 0):

Genome	Weight	Value	Fitness
1 1 1 0 0	17kg	$16	0
0 0 1 0 0	4kg	$10	10
0 1 0 0 1	3kg	$4	4
1 1 0 0 1	15kg	$8	8

Solving Knapsack with a GA

- Ordered by fitness:

Genome	Weight	Value	Fitness
0 0 1 0 0	4kg	$10	10
1 1 0 0 1	15kg	$8	8
0 1 0 0 1	3kg	$4	4
1 1 1 0 0	17kg	$16	0

Solving Knapsack with a GA

- Random selection based on fitness (with replacement):

Genome	Weight	Value	Fitness
0 0 1 0 0	4kg	$10	10
0 0 1 0 0	4kg	$10	10
1 1 0 0 1	15kg	$8	8
0 1 0 0 1	3kg	$4	4

Solving Knapsack with a GA

- Random pairing:

Genome	Weight	Value	Fitness
0 0 1 0 0	4kg	$10	10
1 1 0 0 1	15kg	$8	8
0 0 1 0 0	4kg	$10	10
0 1 0 0 1	3kg	$4	4

Solving Knapsack with a GA

- Crossover (recombination):

0 0 1 | 0 0	0 0 | 1 0 0
1 1 0 | 0 1	0 1 | 0 0 1
↓	↓
0 0 1 | 0 1	0 0 | 0 0 1
1 1 0 | 0 0	0 1 | 1 0 0

Solving Knapsack with a GA

- Results from crossover

Genome	Weight	Value	Fitness
0 0 1 0 1	6kg	$12	12
1 1 0 0 0	13kg	$6	6
0 0 0 0 1	2kg	$2	2
0 1 1 0 0	5kg	$12	12

Solving Knapsack with a GA

- Random mutation

Genome	Weight	Value	Fitness
0 0 1 1 1	7kg	$13	13
1 1 0 0 0	13kg	$6	6
0 0 0 0 0	0kg	$0	0
0 1 1 0 0	5kg	$12	12

Solving Knapsack with a GA

- Population (generation 1):

Genome	Weight	Value	Fitness
0 0 1 1 1	7kg	$13	13
0 1 1 0 0	5kg	$12	12
1 1 0 0 0	13kg	$6	6
0 0 0 0 0	0kg	$0	0

Solving Knapsack with a GA

- Random selection based on fitness (with replacement):

Genome	Weight	Value	Fitness
0 0 1 1 1	7kg	$13	13
0 0 1 1 1	7kg	$13	13
0 1 1 0 0	5kg	$12	12
0 1 1 0 0	5kg	$12	12

Solving Knapsack with a GA

- Random pairing:

Genome	Weight	Value	Fitness
0 0 1 1 1	7kg	$13	13
0 1 1 0 0	5kg	$12	12
0 0 1 1 1	7kg	$13	13
0 1 1 0 0	5kg	$12	12

Solving Knapsack with a GA

- Crossover (recombination):

0 0 1 1 | 1　　　　0 0 | 1 1 1

0 1 1 0 | 0　　　　0 1 | 1 0 0

↓　　　　↓

0 0 1 1 | 0　　　　0 0 | 1 0 0

0 1 1 0 | 1　　　　0 1 | 1 1 1

Solving Knapsack with a GA

- Results from crossover:

Genome	Weight	Value	Fitness
0 0 1 1 0	5kg	$11	11
0 1 1 0 1	7kg	$14	14
0 0 1 0 0	4kg	$10	10
0 1 1 1 1	8kg	$15	15

Solving Knapsack with a GA

- Random mutation:

Genome	Weight	Value	Fitness
0 0 1 1 0	5kg	$11	11
0 1 1 0 0	5kg	$12	12
0 0 1 0 0	4kg	$10	10
0 1 1 1 1	8kg	$15	15

Solving Knapsack with a GA

- Population (generation 2):

Genome	Weight	Value	Fitness
0 1 1 1 1	8kg	$15	15
0 1 1 0 0	5kg	$12	12
0 0 1 1 0	5kg	$11	11
0 0 1 0 0	4kg	$10	10

Solving Knapsack with a GA

- Population (generation 2):

Genome	Weight	Value	Fitness
0 1 1 1 1	8kg	$15	15
0 1 1 0 0	5kg	$12	12
0 0 1 1 0	5kg	$11	11
0 0 1 0 0	4kg	$10	10

Pros and Cons of GAs

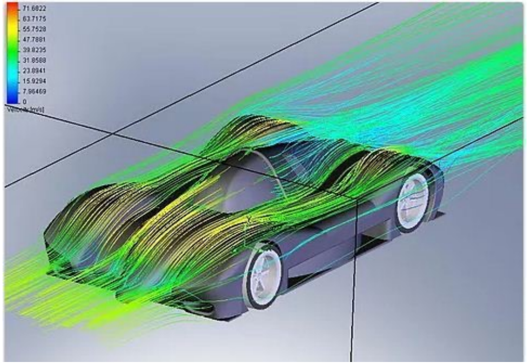
- Pros
  - General approximate optimization strategy
  - Find a good solution quickly in a large space
  - Requires minimal domain-specific knowledge
  - Simple to implement
  - Inherently parallelizable
- Cons
  - No guarantees of performance
  - May get stuck at local maximum
  - May be outperformed by specialized strategies

Applications of GAs

- Traveling Salesman Problem (TSP)
  - Uses specialized encoding and crossover operations
  - Outperformed by specialized approximation strategies on very large instances

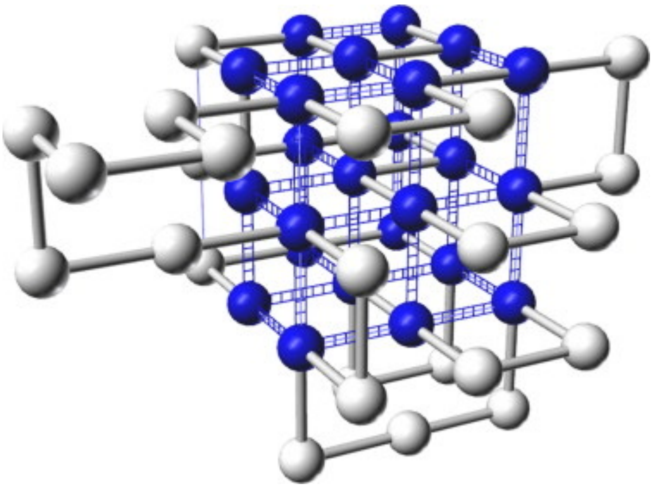
Applications of GAs

- Automotive Design
- Engineering
- Robotics



Applications of GAs

- Molecular structure optimization
- Protein folding prediction



Applications of GAs

- Cryptography
- Financial modeling