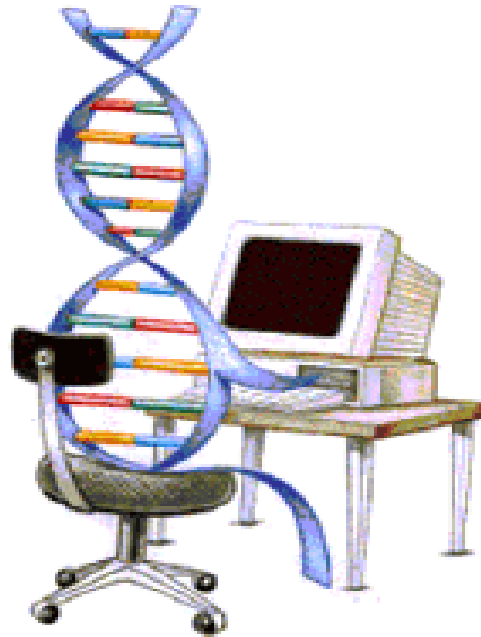

Learning to optimize

Lecture 07.01



Global optimization

- Optimization techniques are typically used in problems that have **many possible solutions across many variables**, and that need to find the best solution over the combinations of all variables
 - The goal is to find the best combination of the variables by trying many different solutions and scoring them to determine their quality
 - Optimization is typically used in cases where **there are too many possible solutions to try them all**
-

When to try optimizations

- The primary requirements for applying optimization techniques:
 - The problem has a **defined cost function**
 - **Similar solutions tend to yield similar cost**
 - Not every problem with these properties will be solvable with optimization, but there's a good chance that optimization will return some interesting results that you hadn't considered.
-

First example (as always): *Hello world*

- Disclaimer: this particular “optimization” is here just to present all different optimization techniques – it is not a real application
 - We want to “optimize” a random string into “Hello world”
 - We cannot do exhaustive search among 52^{11} strings to find the best which matches our goal
-

Setup

- To restate this as an optimization problem:
 - Represent each solution as a sequence of numbers (each character is already a number!)
 - Have a way to evaluate the quality of the solution: in this case we use *Hamming distance** between the solution and the target string ‘Hello world’

**Hamming distance* between 2 strings is the total number of positions at which characters in 2 strings do not match

Random “optimization”

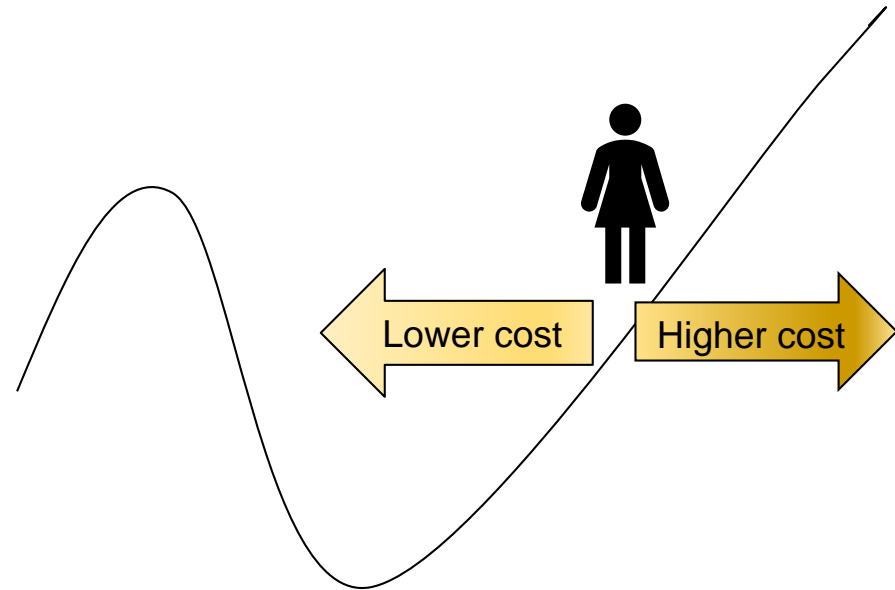
- Randomly trying different solutions is very inefficient because the probability of getting anything close to perfect *Hello world* is $(1/52)^{11}$
 - This “optimization” does not make use of a better solution that has already been found, it just tries another random guess
 - No matter how many random guesses you try – the fitness of the resulting function is very low (the distance from the target remains high)
-

Hill climbing

- An alternate method of random searching is called *hill climbing*.
 - Hill climbing starts with a random solution and looks at the set of neighboring solutions for those that are better (have a better value for fitness function).
-

Seeking a lower cost on the hill

- You have been randomly dropped into this landscape
- You want to reach the lowest point to find water
- You look in each direction and walk toward wherever the land slopes downward
- You continue to walk in the most steeply sloping direction until you reach a point where the terrain is flat or begins sloping uphill



Hello world: hill climbing

- Pick a random solution
- Create list of neighbors trying to change one letter at a time

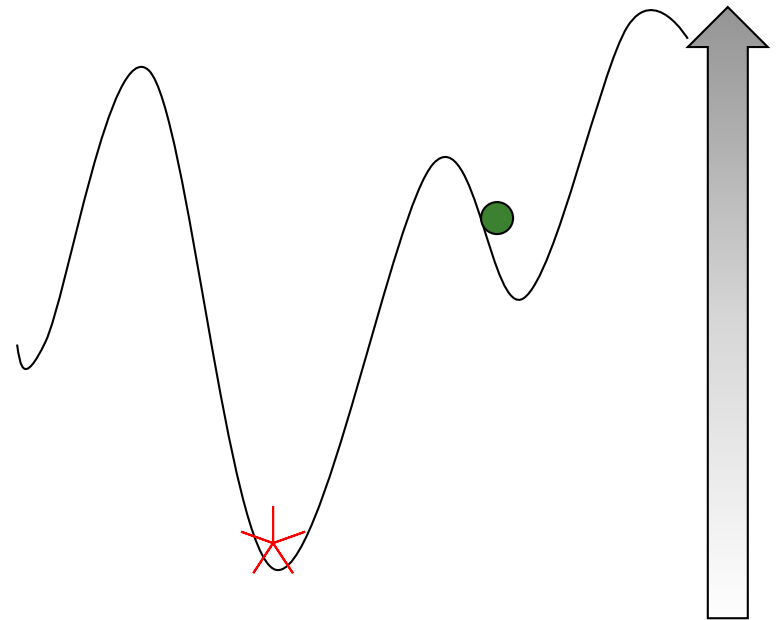
```
# Create list of neighboring solutions  
neighbors = []
```

```
for j in range(len(sol)):  
    # One away in each direction  
    neighbors.append(sol[0:j] + chr(ord(sol[j]) + 1) + sol[j + 1:])  
    neighbors.append(sol[0:j] + chr(ord(sol[j]) - 1) + sol[j + 1:])
```

- See if any of them is better than current best
 - Continue working with this better solution (moving into the direction of decreased cost – better solution fitness)
-

Hill climbing: good and bad

- Hill climbing runs quickly
- Usually finds a better solution than random searching
- One major drawback: the final solution is a *local minimum*, a solution better than those around it but not the best overall
- The best overall is called the *global minimum*, which is our ultimate goal



Borrowing from physics: annealing

- Annealing is the process of heating up an alloy to extremely high temperature and then cooling it down slowly
 - The atoms first jump around a lot and then gradually settle into a low energy state, finding a low energy configuration
-

Simulated annealing

- Pick a random solution
 - Set up the temperature, which starts very high and gradually gets lower
 - In each iteration, one of the numbers in the solution is randomly chosen and changed in a certain direction
 - Main idea:
 - If the new cost is *lower*, the new solution becomes the current solution (as before)
 - If the cost is *higher*, the new solution **may still become the current solution** with a certain probability - this may help to get out of the local minimum
-

Probability of moving in the wrong direction gradually “cools down”

- In some cases, it's necessary to move to a worse solution before you can get to a better one.
 - Simulated annealing improves the final result because it will almost always accept a move for the better, and because it is willing to accept a worse solution near the beginning of the process.
 - As the process goes on, the algorithm becomes less and less likely to accept a worse solution, until at the end it will only accept a better solution.
-

$$p = e^{(-highcost - lowcost) / temperature}$$

- The probability of a higher-cost solution being accepted is given by this formula
 - Temperature is decreasing with each iteration
 - Since the temperature (the willingness to accept a worse solution) starts very high, the exponent will at first be close to 0, so the probability will almost be 1
 - As the temperature decreases, the difference between the high cost and the low cost becomes more important—a bigger difference leads to a lower probability, so the algorithm will favor only slightly worse solutions over much worse ones
-

**BORROWING FROM BIOLOGY:
GENETIC ALGORITHM OPTIMIZATION**

History Of Genetic Algorithms

- “Evolutionary Computing” was introduced in the 1960s by **I. Rechenberg**.
 - John Holland wrote the first book on Genetic Algorithms ‘**Adaptation in Natural and Artificial Systems**’ in 1975.
 - In 1992 **John Koza** used genetic algorithm to **evolve programs** to perform certain tasks. He called his method “**Genetic Programming**”.
-

What Are Genetic Algorithms (GAs)?

Genetic Algorithms are *search* and *optimization* techniques based on Darwin's Principle of *Natural Selection*.

Why GAs

- Evolution is known to be a successful, robust method to produce adaptations (solutions) to different environments (problems)
 - GAs can search a very large space of hypotheses containing complex interacting parts
 - GAs are inherently parallelizable
-

Darwin's Principle Of Natural Selection

- IF** there are organisms that reproduce, and
- IF** offspring inherit traits from their progenitors, and
- IF** there is variability of traits, and
- IF** the environment cannot support all members of a growing population,
- THEN** those members of the population with less-adaptive traits (determined by the environment) will die out, and
- THEN** those members with more-adaptive traits (determined by the environment) will thrive

The result is the evolution of **species**.

Basic Idea of Natural Selection

“Select The Best, Discard The Rest”

An Example of Natural Selection

- **Giraffes have long necks.**

Giraffes with slightly longer necks could feed on leaves of higher branches when all lower ones had been eaten off.

→ They had a better chance of survival.

→ Favorable characteristic propagated through generations of giraffes.

→ Now, evolved species has long necks.

NOTE: Longer necks may have been a deviant characteristic (**mutation**) initially but since it was favorable, was propagated over generations. Now an established trait.

So, some mutations are beneficial.

Evolution Through Natural Selection

Initial Population Of Animals



Struggle For Existence-Survival Of the Fittest

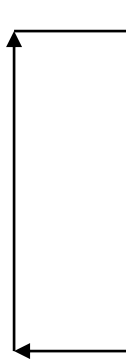


Surviving Individuals Reproduce, Propagate Favorable Characteristics



Evolved Species

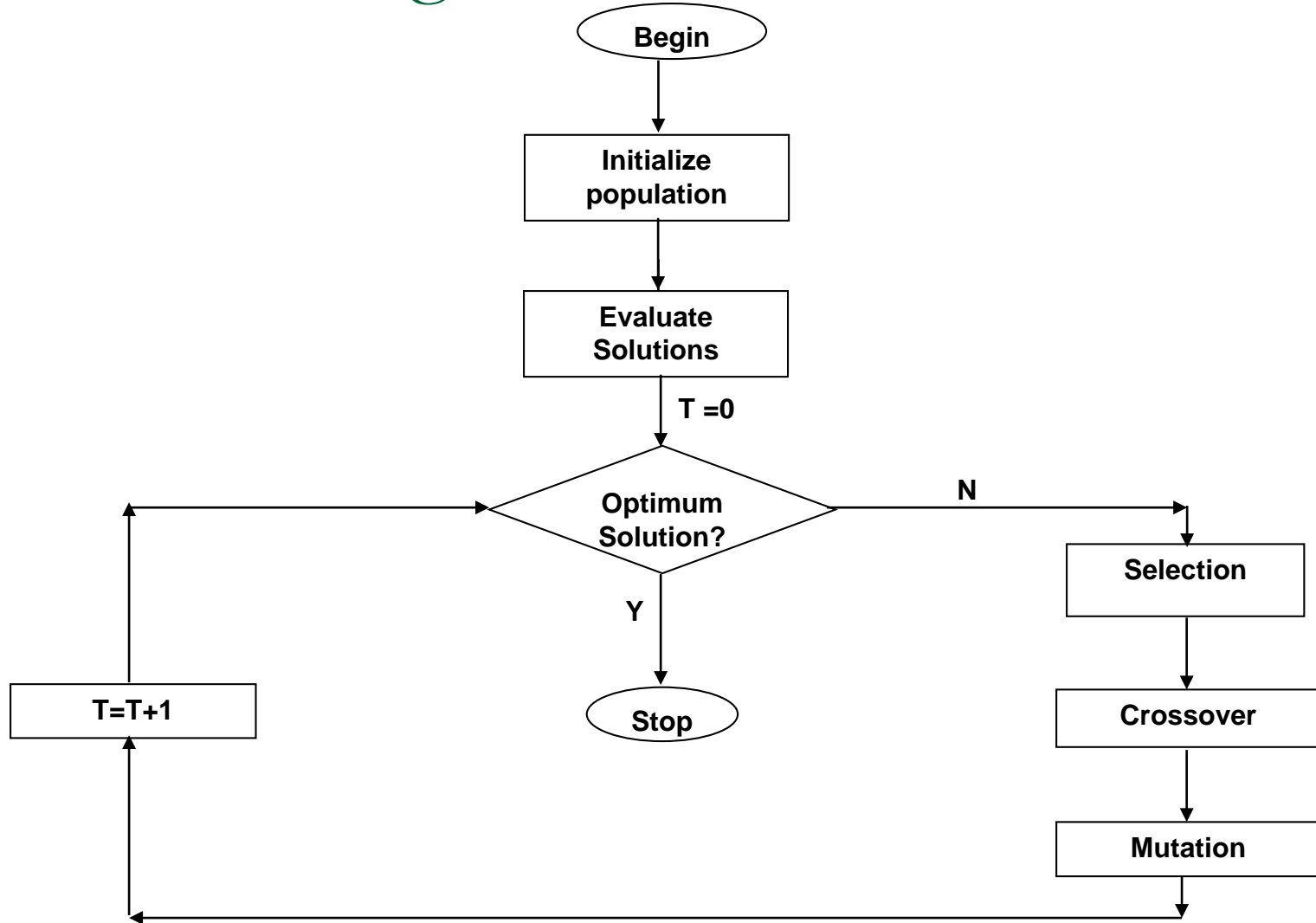
(Favorable Characteristic Now a Trait Of Species)



Millions Of Years

Genetic Algorithms Implement
Optimization Strategies By **Simulating
Evolution Of Species** Through Natural
Selection.

Working Mechanism Of GAs



Mapping Nature to an Algorithm

Nature	Computer
Population	Set of initial hypotheses (possible solutions)
Individual	Solution to a problem
Fitness	Quality of a solution
Chromosome	Encoding for a Solution
Gene	Part of the encoding of a solution
Mating (crossover)	Crossover

GA design

-
1. *Encoding*
 2. *Fitness function*
 3. *Selection*
 4. *Mating (crossover)*
 5. *Mutation*

1. Encoding

*The process of representing the solution in the form of a **string** that conveys the necessary information.*

- Just as in a chromosome, each gene controls a particular characteristic of the individual, similarly, each bit in the string represents a characteristic of the solution.
-

Encoding Methods 1/3

- **Binary Encoding** – Most common method of encoding. Chromosomes are strings of 1s and 0s and each position in the chromosome represents a particular characteristic of the problem.

Chromosome A	10110010110011100101
Chromosome B	11111110000000011111

Sample Problem

The **Traveling Salesman Problem** is defined as:

‘We are given a set of cities and a symmetric distance matrix that indicates the cost of travel from each city to every other city.

*The goal is to find **the shortest circular tour**, visiting every city exactly once, so as to **minimize the total travel cost**, which includes the cost of traveling from the last city back to the first city’.*

Encoding Methods 2/3

- **Permutation Encoding** – Useful in scheduling problems such as the Traveling Salesman Problem (TSP). Example. In TSP, every chromosome is a string of numbers, each of which represents a city to be visited in this order.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Encoding Methods 3/3

- **Value Encoding** – Used in problems where complicated values, such as real numbers, are used and where binary encoding would not suffice.
Good for some problems, but *often necessary to develop some specific crossover and mutation techniques for these chromosomes.*

Chromosome A	1.235 5.323 0.454 2.321 2.454
Chromosome B	(left), (back), (left), (right), (forward)

2. Fitness Function

A fitness function quantifies the optimality of a solution (chromosome) so that that particular solution may be ranked against all the other solutions.

- A fitness value is assigned to each solution depending on how close it is to the perfect solution.
 - Ideal fitness function correlates closely to goal + is quickly computable.
 - Example. In TSP, $f(x)$ is sum of distances between the cities in solution. The lesser the value, the fitter the solution is.
-

3. Selection

The process that determines which solutions are to be preserved and allowed to reproduce and which ones deserve to die out.

- The primary objective of the selection operator is to **emphasize the good solutions** and **eliminate the bad solutions** in a population, **while keeping the population size constant**.
 - “Selects The Best, Discards The Rest”.
-

Elite Selection

- Sort solutions by fitness (descending).
 - Make multiple copies of the top solutions (parthenogenesis – cloning).
 - Eliminate bad solutions from the population so that multiple copies of good solutions can be placed in the population.
-

Roulette Wheel Selection

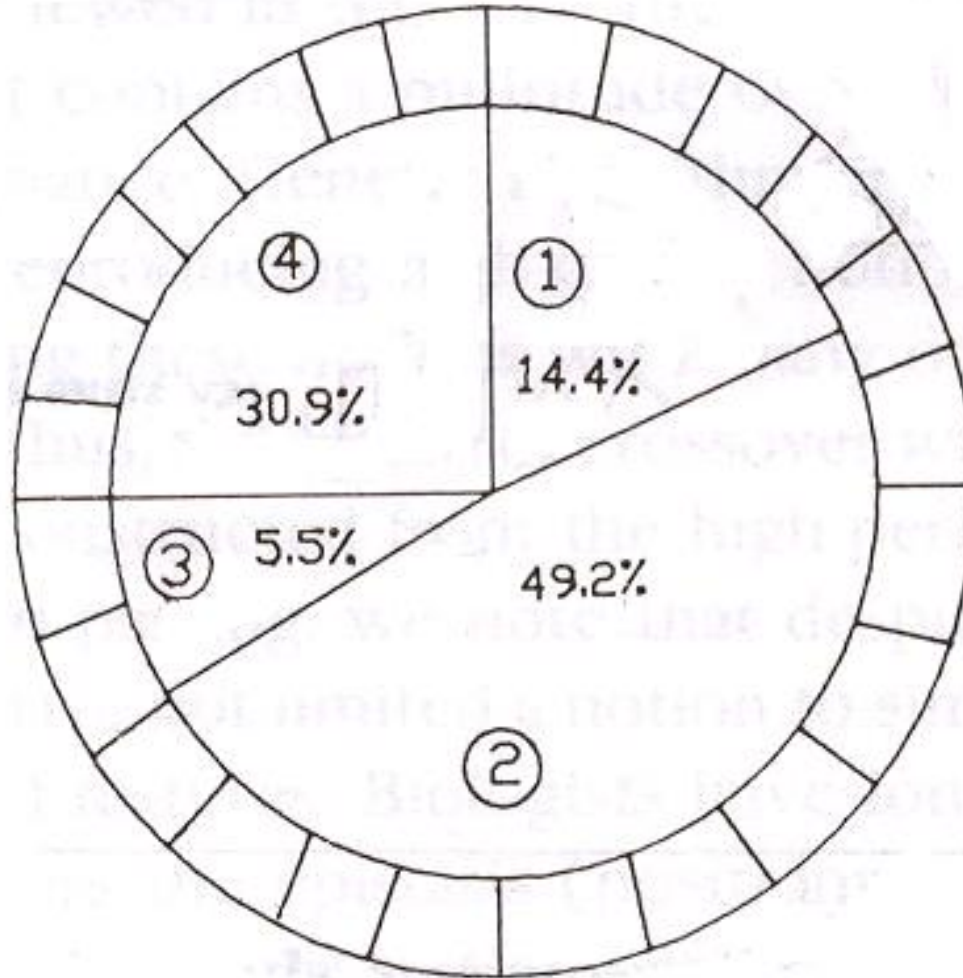
- Each current string in the population has a slot assigned to it which is in **proportion to its fitness**.
- We spin the weighted *roulette wheel* thus defined n times (where n is the total population size).
- Each time Roulette Wheel stops, the string corresponding to that slot is selected for the next generation.

Strings that are fitter are assigned a larger slot and hence have a better chance of appearing in the new population.

Example Of Roulette Wheel Selection

No.	String	Fitness	% Of Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

Roulette Wheel

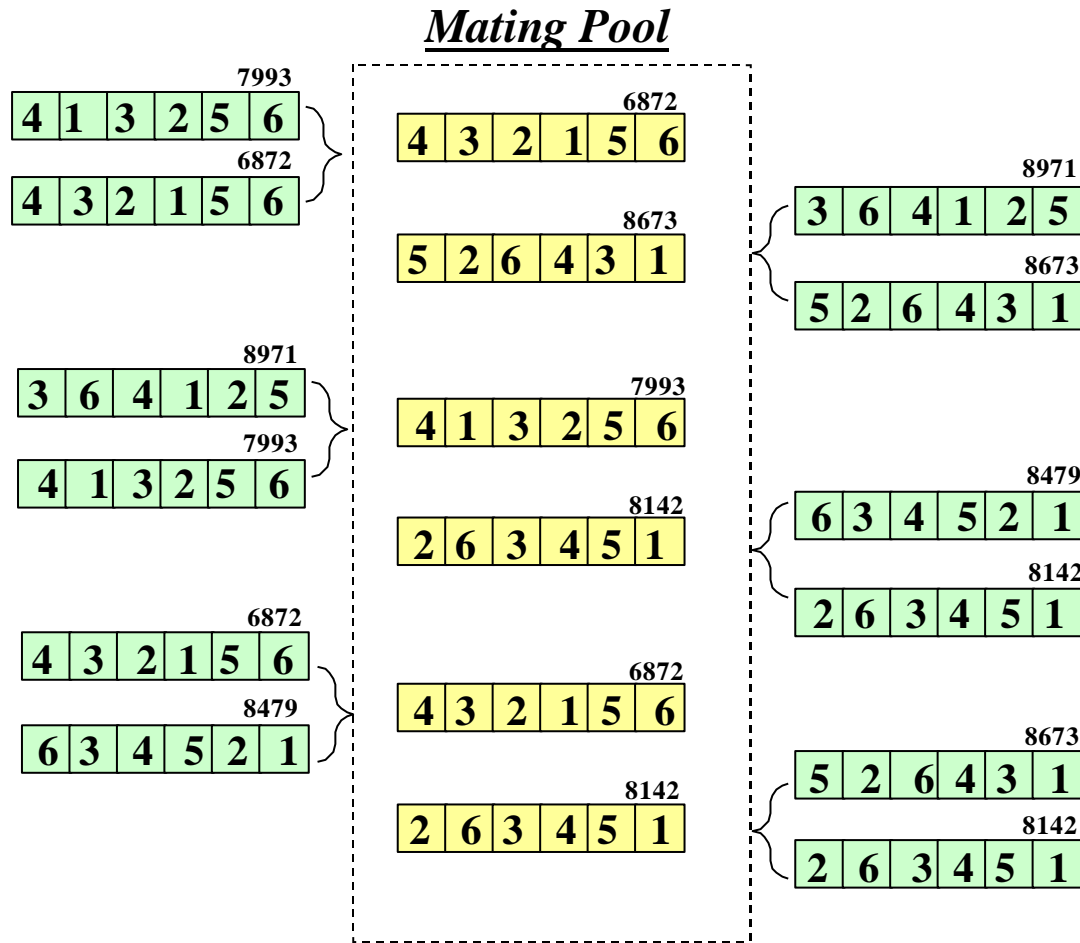


Tournament selection

- **Tournament Selection:** Two members are chosen at random from a population.
 - With some predefined probability p the more fit of these two is then selected, and with probability $1-p$ the less fit hypothesis is selected.

Sometimes TS yields a more diverse population than RS.

Tournament Selection Example



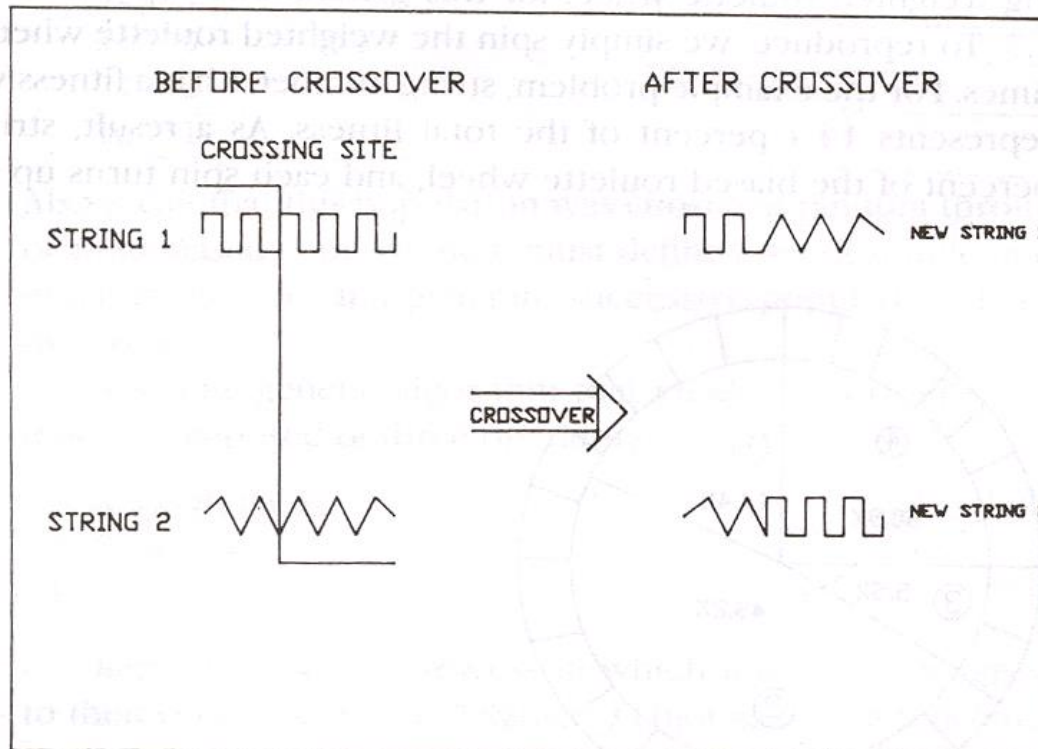
4. Mating (Crossover)

It is the process in which two chromosomes (encodings) combine their genetic material (bits) to produce a new offspring which possesses a mix of their characteristics.

- *Two strings are picked from the mating pool (or from elite) at random to cross over.*
 - *The method chosen depends on the Encoding Method.*
-

Crossover Methods 1/3

- **Single Point Crossover-** A random point is chosen on the individual chromosomes (strings) and the genetic material is exchanged at this point.



Single Point Crossover

Chromosome1	11011 00100110110
Chromosome 2	11011 11000011110
Offspring 1	11011 11000011110
Offspring 2	11011 00100110110

Crossover Methods 2/3

- **Two-Point Crossover-** Two random points are chosen on the individual chromosomes (strings) and the genetic material is exchanged at these points.

Chromosome1	11011 00100 110110
Chromosome 2	10101 11000 011110
Offspring 1	10101 00100 011110
Offspring 2	11011 11000 110110

NOTE: These chromosomes are different from the last example.

Crossover Methods 3/3

- **Uniform Crossover-** Each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes.

Chromosome1	11011 00100 110110
Chromosome 2	10101 11000 011110
Offspring	10111 00000 110110

NOTE: Uniform Crossover yields ONLY 1 offspring.

Crossover summary

- Crossover between 2 good solutions **MAY NOT ALWAYS** yield a better or as good a solution.
 - Since parents are good, probability of the child being good is high.
 - If offspring is not good (poor solution), it will be removed in the next iteration during “Selection”.
-

Elitism in mating

Elitism is a method which copies the best chromosome to the new offspring population before crossover and mutation.

- When creating a new population by crossover or mutation the best chromosome might be lost.
 - Forces GAs to retain some number of the best individuals at each generation.
 - Has been found that elitism significantly improves performance.
-

5. Mutation

It is the process by which a string is deliberately changed at random to maintain diversity in the population set.

We saw in the giraffes' example, that mutations could be beneficial.

Mutation Probability- determines how often the parts of a chromosome will be mutated.

Example Of Mutation

- For chromosomes using Binary Encoding, randomly selected bits are inverted.

Offspring	11011 00100 110110
Mutated Offspring	11010 00100 100110

NOTE: The number of bits to be inverted depends on the Mutation Probability.

GA for evolving strings

1. Encoding: string itself
2. Fitness function: Hamming distance

```
fitness_function = lambda x: sum([abs(ord(x[i]) -  
                                ord(TARGET_STRING[i]))  
                                for i in range(target_length)])
```

3. Selection: elitism

In each generation start a new population from top-fit individuals

4. Mating: single-point crossover

```
def string_crossover(p1, p2, params=None):  
    ipos = random.randint(1, len(p1) - 2)  
    return p1[0:ipos] + p2[ipos:]
```

5. Mutation: point mutation

Replace character at random position by a random valid character

To see the results run [hello_world.py](#)

Sample application 1

FINDING BEST CLASSIFICATION RULES

Sample code: <https://github.com/andresoSw/gabil-ga>

GABIL [DeJong et al. 1993]

- Learn a set of rules

Representation:

- Each hypothesis is a set of rules
- To represent a set of rules, the bit-string representation of individual rules are concatenated

Example

IF $a_1 = T$ AND $a_2 = F$ THEN $c = T$;

IF $a_2 = T$ THEN $c = F$

a_1 a_2 c a_1 a_2 c

10 01 1 10 11 0

Representing Hypotheses (Binary encoding)

Represent

$(\text{Outlook} = \text{Overcast} \vee \text{Rain}) \wedge (\text{Wind} = \text{Strong})$

by

Outlook	Wind
011	10

Represent

$\text{IF Wind} = \text{Strong THEN PlayTennis} = \text{yes}$

by

Outlook	Wind	PlayTennis
111	10	10

Means
Outlook=any.

Fitness function for a set of rules

Fitness function:

$$\text{Fitness}(h) = (\text{correct}(h))^2$$

$\text{correct}(h)$: the percent of all training examples correctly classified

Mutation and crossover

- Use the standard mutation operator
 - Crossover: extension of the two-point crossover operator
 - want variable-length rule sets
 - want only well-formed bitstring hypotheses
 - Crossover with variable-length bitstrings
 1. choose two crossover points for h_1 . Let d_1 (d_2) be the distance to the rule boundary immediately to its left.
 2. now restrict points in h_2 to those that have the same d_1 and d_2 value
-

Crossover example

	a_1	a_2	c	a_1	a_2	c
h1 :	10	01	1	11	10	0
h2 :	01	11	0	10	01	0

Suppose crossover points for h1, are after bits 1, 8
($d1=1;d2=3$)

	a_1	a_2	c	a_1	a_2	c
h1 :	1	[0	01	1	11	1]0 0

Allowed pairs of crossover points for h2 are $\langle 1;3 \rangle$, $\langle 1;8 \rangle$,
 $\langle 6;8 \rangle$.

If pair $\langle 1;3 \rangle$ is chosen,

	a_1	a_2	c	a_1	a_2	c
h2 :	0	[1	1]1	0	10	01 0

the result is:

Crossover example (cont.)

	a_1	a_2	c		a_1	a_2	c
h1 :	<u>1</u>	0	1		1	<u>1</u>	<u>0</u>
h2 :	0	<u>1</u>	1		1	0	0

	a_1	a_2	c
c1 :	<u>1</u>	<u>1</u>	<u>0</u>

	a_1	a_2	c		a_1	a_2	c		a_1	a_2	c
c2 :	0	0	1		1	1	0		1	0	0

Sample application 2

GROUP TRAVEL OPTIMIZATION

Setup

- Family members meet up in New York.
- They arrive on the same day and leave on the same day, and they want to share transportation to and from the airport.
- There are about 9 possible flights per day for each family member
- Flights are arriving-leaving at different times.
- The flights also vary in price and in duration.

```
people = [('John', 'BOS'),  
          ('Mary', 'DAL'),  
          ('Laura', 'CAK'),  
          ('Abe', 'MIA'),  
          ('Greg', 'ORD'),  
          ('Lee', 'OMA')]
```

```
# LaGuardia airport in New York  
destination='LGA'
```

```
LGA, MIA, 20:27, 23:42, 169  
MIA, LGA, 19:53, 22:21, 173  
LGA, BOS, 6:39, 8:09, 86  
BOS, LGA, 6:17, 8:26, 89  
LGA, BOS, 8:23, 10:28, 149  
...
```

Flights in file schedule.txt

Representing solutions

- We represent each possible solution as a list of numbers.
 - Each number represents the position of a flight in a list of flights sorted by time.
 - Since each person needs an outbound flight and a return flight, the length of this list is twice the number of people.
-

Sample solution

```
[1, 4, 3, 2, 7, 3, 6, 3, 2, 4, 5, 3]
```

John Mary

- John takes the second flight of the day from Boston to New York, and the fifth flight back to Boston on the day he returns.
- Mary takes the fourth flight from Dallas to New York, and the third flight back.

```
people = [('John', 'BOS'),  
          ('Mary', 'DAL'),  
          ('Laura', 'CAK'),  
          ('Abe', 'MIA'),  
          ('Greg', 'ORD'),  
          ('Lee', 'OMA')]
```

Fitness function design: most difficult part

- The goal is to find a set of flights that minimizes the cost function.
 - The cost function has to return a value that represents how bad a solution is.
 - There is no particular scale for badness: the only requirement is that the function returns larger values for worse solutions.
-

What to include into the fitness function

- **Price**
 - The total price of all the plane tickets [possibly a weighted average that takes financial situations into account].
- **Travel time**
 - The total time that everyone must spend on a plane.
- **Waiting time**
 - Time spent at the airport waiting for the other members of the party to arrive.
- **Departure time**
 - Flights that leave too early in the morning may impose an additional cost by requiring travelers to miss out on sleep.
- **Car rental period**
 - If the party rents a car, they must return it earlier in the day, or be forced to pay for a whole extra day.

See `schedule_cost` function in file *travel.py*

Run schedule optimization

- Execute *travel.py*

How much better is the GA solution comparing to the random optimizer?

Sample application 3

STUDENTS TO DORMS ASSIGNMENT

Student Dorm Optimization

- The goal is to assign students to dorms depending on their first and second choices.
 - Although this is a very specific example, it's easy to generalize it to other problems—the exact same code can be used to:
 - assign tables to players in an online card game
 - assign bugs to developers in a large coding project
 - even to assign housework to household members.
 - In all these problems the purpose is to take preferences from individuals and produce the overall optimal result.
-

Dorm preferences

```
# The dorms, each of which has two available spaces  
dorms=[ 'Dolliver', 'Crosby', 'Hill', 'Carriage', 'MODs' ]
```

```
prefs=[ ('Daniel', ('Carriage', 'Hill')),  
        ('Steven', ('Dolliver', 'MODs')),  
        ('Akarsh', ('Crosby', 'Dolliver')),  
        ('Kelvin', ('Dolliver', 'MODs')),  
        ('Betty', ('Crosby', 'Carriage')),  
        ('Jeff', ('Hill', 'MODs')),  
        ('Cat', ('MODs', 'Crosby')),  
        ('Michael', ('Carriage', 'Hill')),  
        ('Gary', ('Carriage', 'Hill')),  
        ('James', ('Hill', 'Crosby'))]
```

- As you see, the perfect assignment is impossible: 3 students want Carriage, but each dorm has only 2 available spaces

Representing solutions

```
# The dorms, each of which has two available spaces  
dorms= ['Dolliver', 'Crosby', 'Hill', 'Carriage', 'MODs']
```

- In theory we could create a list of numbers, one for each student, where each number represents the dorm in which you put the student.
- The problem is that this representation doesn't constrain the solution to only two students in each dorm.
- A list of all zeros would indicate that everyone had been placed in Dolliver, which isn't a real solution at all.
- We could potentially

Considering only valid solutions

- In general, it's better not to waste time searching among invalid solutions.
- We need to find a way to represent solutions so that every solution is valid.
- A valid solution is not necessarily a good solution; it just means that there are exactly two students assigned to each dorm.
- We think of every dorm as having two slots, so that there are ten slots in total.
- Each student, in order, is assigned to one of the open slots—the first person can be placed in any one of the ten, the second person can be placed in any of the nine remaining slots, and so on.

Solution is represented as a list of assignments

```
[0,0,0,0,0,0,0,0,0,0]
```

- The *print_solution* in *dorm.py* illustrates how this solution representation works.
- This function first creates a list of slots, two for each dorm. It then loops over every number in the solution and finds the dorm number at that location in the slots list, which is the dorm that a student tries to be assigned to.
- It prints the student and the dorm, and then it removes that slot from the list so no other student will be given that slot.
- After the final iteration, the slots list is empty and every student and dorm assignment has been printed.

Each slot has its own domain

`[0,0,0,0,0,0,0,0,0,0]`

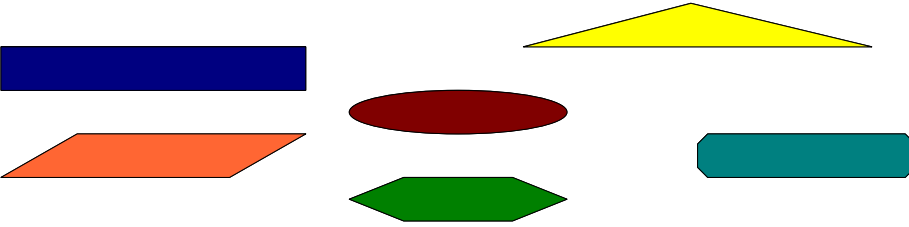
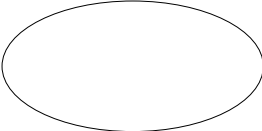

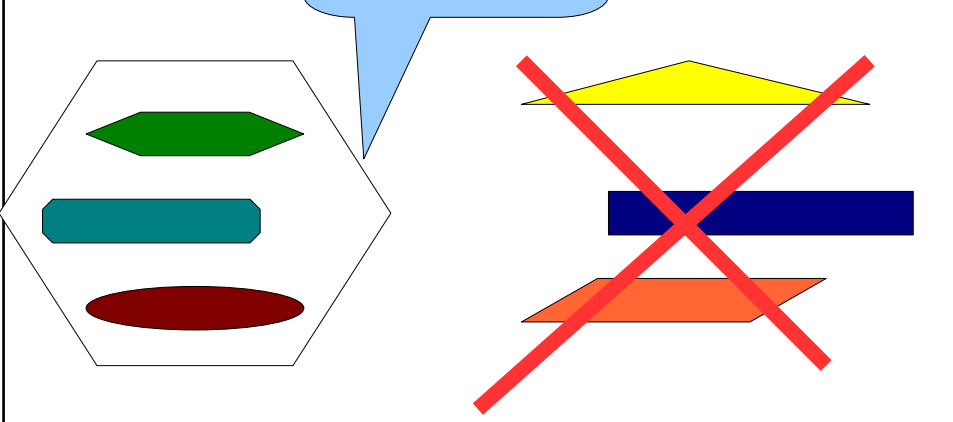
- If you change the numbers to view different solutions, remember that each number must stay in the appropriate range.
 - The first item in the list can be between 0 and 9, the second between 0 and 8, etc.
 - Since the optimization functions will keep the numbers in the ranges specified in the domain parameter, this won't be a problem when optimizing.
-

Fitness function

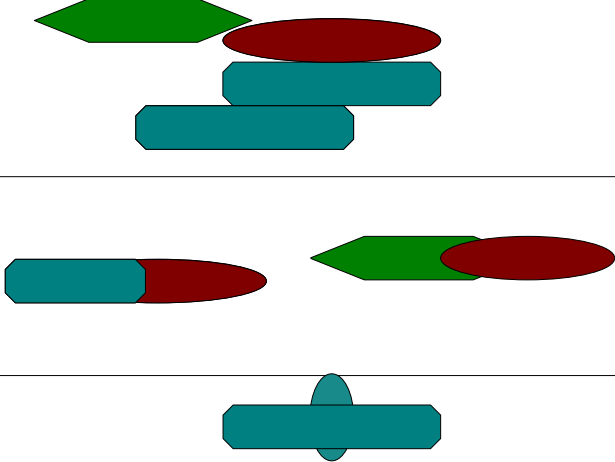
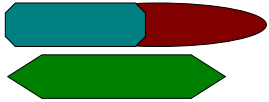
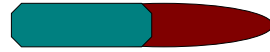
- The cost function works similarly to the print function.
- A list of all slots is constructed and slots are removed as they are used up.
- The cost is calculated by comparing a student's current dorm assignment to his top two choices.
 - Add 0 if the student is currently assigned to his top choice
 - Add 1 if he is assigned to his second choice
 - Add 3 if he is not assigned to either of his choices

GA Summary

Genetic algorithm steps

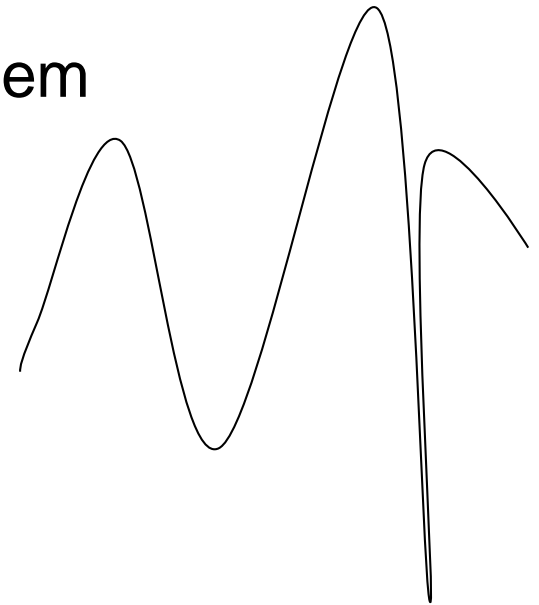
1.	Create population of random individuals	
2.	Choose fitness function : to evaluate how good is a particular individual for a specific purpose defined by a specific problem	
3.	Run several iterations (generations)	
		

Genetic algorithm steps (cont.)

5.	<p>The next generation consists of: Unchanged elite (parthenogenesis)</p> <p>Individuals which combine features of 2 elite parents (mating)</p> <p>Small part of elite individuals changed by random mutation</p>	
6.	<p>Repeat steps 4, 5 until no more significant improvement in the fitness of elite is observed</p>	
	<p>Get the best solution</p>	

When to use Genetic Algorithms

- If you are trying to minimize (maximize) some function $f(x)$ over all values of variables x in \mathbf{X}
- When examining every possible combination of x in \mathbf{X} is infeasible
- When you are concerned with the problem of local minimum (maximum) – random mutation can get you out of trap



Advantages Of GAs over other optimization methods

- **Global Search Methods:** search for the function optimum starting from a *population of points* of the function domain, not a single point.
 - This makes GAs global search methods:
 - They can climb many peaks in parallel, reducing the probability of finding local minima, which is one of the drawbacks of traditional optimization methods.
 - **GAs can be easily used in *parallel machines***
 - Since most computational time is spent in evaluating a solution, with multiple processors all solutions in a population can be evaluated in a distributed manner.
-

GA applications

GAs are highly effective in searching a **large, poorly defined search space** even in the presence of high-dimensionality, multi-modality, discontinuity and noise.

Success stories:

- ❑ Finding which concert hall shape gives the best acoustics
 - ❑ Designing an optimal wing for a supersonic aircraft
 - ❑ Suggesting the best library of chemicals to research as potential drugs
 - ❑ Automatically designing a chip for voice recognition
-

Think

- Suggest optimization problems which can be efficiently solved with genetic algorithm

