

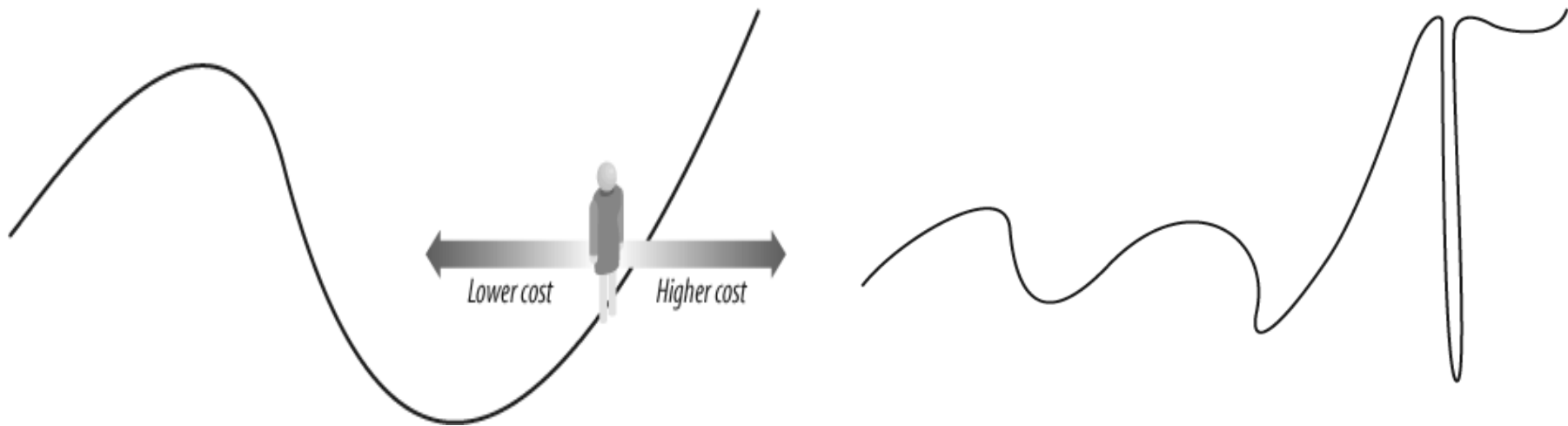
Genetic Algorithm in Python

Data mining lab 6

When to use genetic algorithms

John Holland (1975)

- Optimization: minimize (maximize) some function $f(x)$ over all possible values of variables x in X
- A brute force: examining every possible combination of x in X in order to determine the element for which f is optimal: infeasible
- Optimization techniques are heuristic.
- The problem of local maximum (minimum).



Mutation introduces randomness in the method to get out of trap

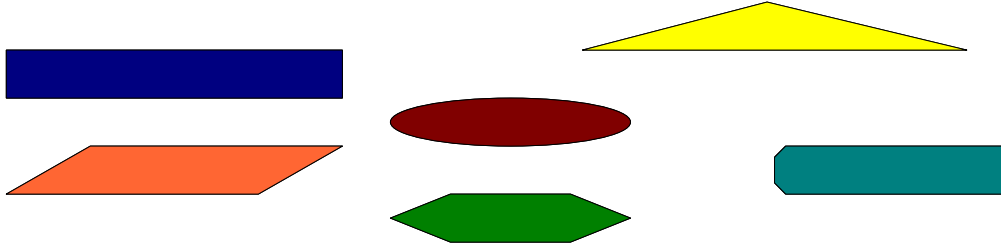
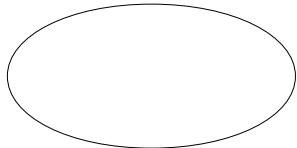
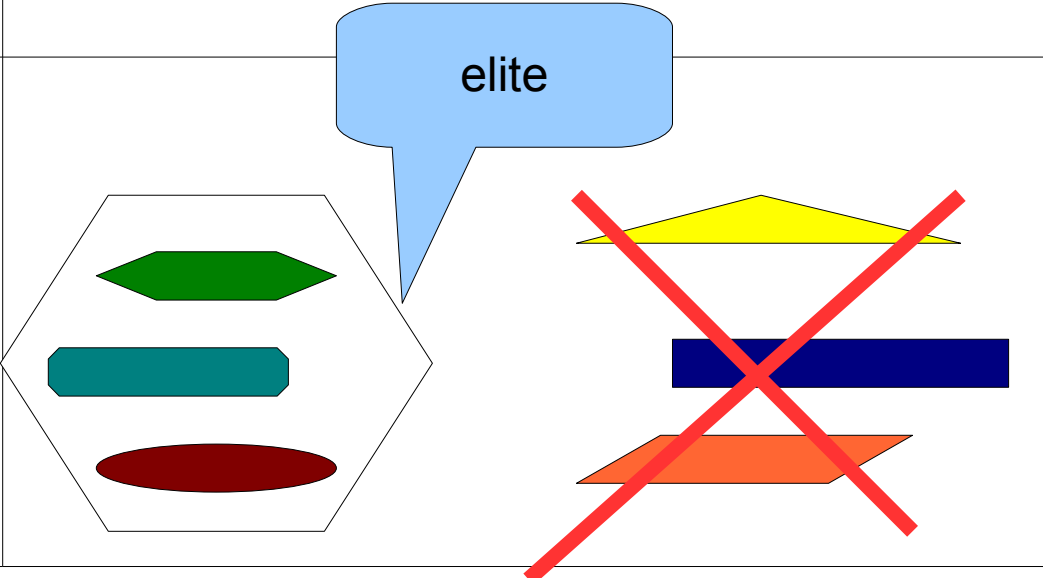
Evolution

- There is a population of individuals with randomly chosen values of variables (features)
- There are some environmental conditions which demand from an individual to have certain features
- The individuals which have the features which are best suited for these conditions have advantage over other individuals, they survive till the reproductive age and reproduce

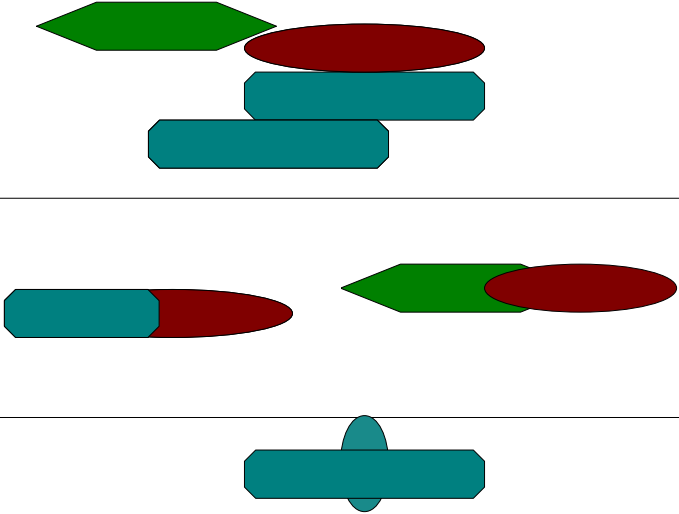


Variation – the pool for the evolution

- The best suited individuals (the fittest) survive, reproduce and mix their features with other surviving individuals
- In the simplest model, they contribute part of the features to a new individual in the next generation, and another part comes from a second parent
- The new individual can undergo the process of mutation – random change of one of his features. This occurs rarely.

Genetic algorithm: the main steps I

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: the main steps II

5.	<p>The next generation consists of: Unchanged elite (parthenogenesis)</p> <p>Individuals which combine features of 2 elite parents (recombinant)</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>	
		

"Hello World" program for genetic algorithms

- Simple example: random population of strings evolves into a predefined template "Hello World"
- For simplicity:
 - random strings have the same length as the target string
 - Fitness function is calculated as the closeness of the given string to the target string

Fitness function

```
def string_fitness (individual):  
    fitness=0  
    for ipos in range (0,target_length):  
        fitness+=abs( ord(individual[ipos])  
                    - ord (TARGET_STRING[ipos]) )  
    return fitness
```

Basically, all this does it goes through each member of the population and compares it with the target string. It adds up the differences between the characters and uses the cumulative sum as the fitness value (therefore, the lower the value, the better).

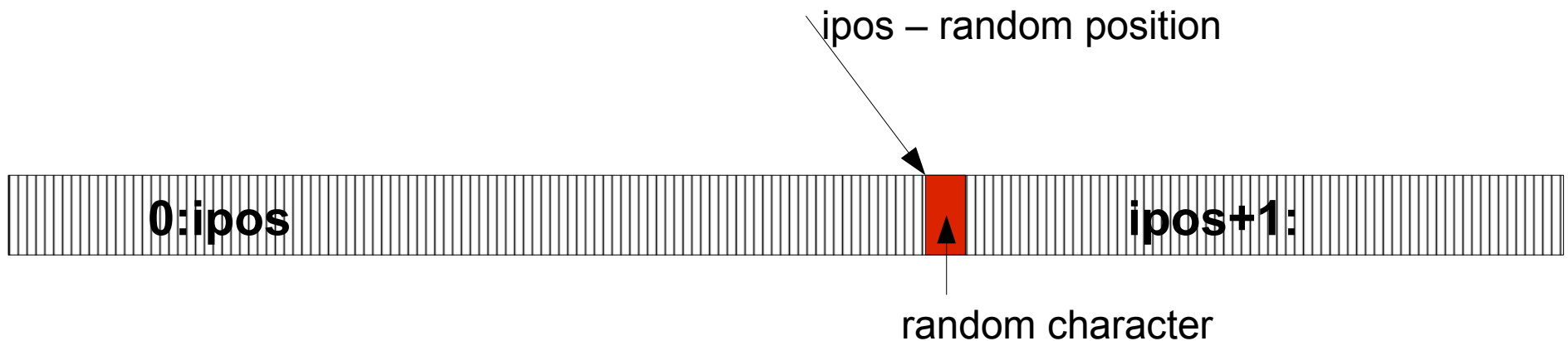
For comparison: random optimizer

```
from ga_helloworld import *  
string_population=init_strings_population(204800)  
best_rand=randomoptimize( string_population,  
                           string_fitness )  
  
print best_rand[1]  
print " score = %d" % best_rand[0]
```

- Random searching isn't a very good optimization method, but it makes it easy to understand exactly what all the algorithms are trying to do, and it also serves as a baseline so you can see if the other algorithms are doing a good job.
- The random optimizer in `random_optimize.py` randomly generates 202,800 random guesses and applies a fitness function for each guess. It keeps track of the best guess (the one with the lowest cost) and returns it.

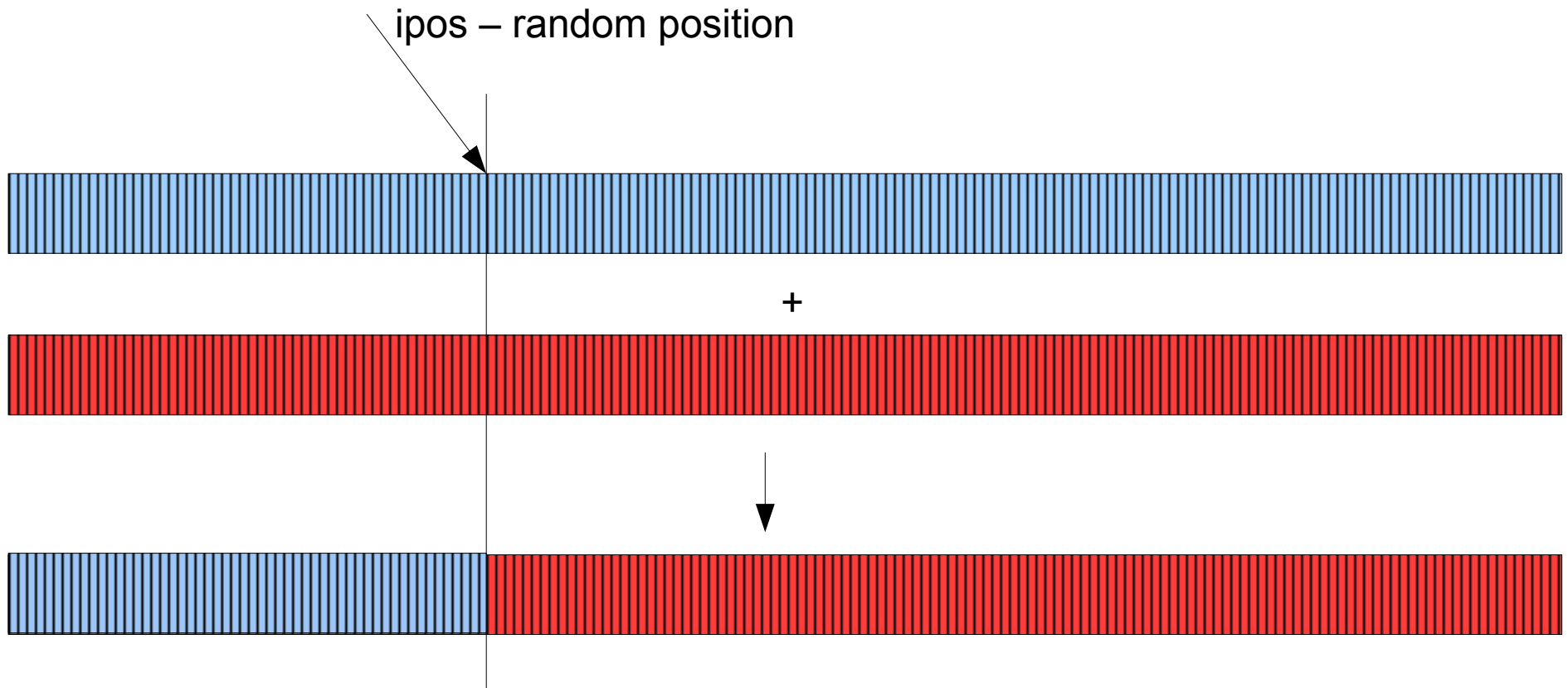
Mutation operation for GA

```
def mutate_string(individual):  
    ipos=random.randint(0,target_length-1)  
    #mutation changes character at random to any available  
        ASCII character from 32 (space) to 90 (Z)  
    rchar=chr(random.randint(0,32000)%90 + 32)  
    individual=individual[0:ipos]+rchar+individual[(ipos+1):]  
    return individual
```



Mate operation (crossover) for GA

```
def string_crossover(p1,p2):  
    ipos=random.randint(1,target_length-2)  
    return p1[0:ipos]+p2[ipos:]
```



Genetic algorithm I

```
def genetic_optimize(population,fitness_function,mutation_function,
    mate_function, mutation_probability, elite, maxiterations):
# How many winners from each generation?

    original_population_size=len(population)

    top_elite=int(elite*original_population_size)

# Main loop

    for i in range(maxiterations):

        individual_scores=[(fitness_function(v),v)
                                for v in population]

        individual_scores.sort( )

        ranked_individuals=[v for (s,v) in individual_scores]

        # Start with the pure winners

        population=ranked_individuals[0:top_elite]
```

Genetic algorithm II

```
# Add mutated and bred forms of the winners

while len(population)<original_population_size:

    if random.random( )<mutation_probability:

        # Mutation

        c=random.randint(0,top_elite)

        population.append( mutation_function

                            (ranked_individuals[c]))

    else:

        # Crossover

        c1=random.randint(0,top_elite)

        c2=random.randint(0,top_elite)

if individual_scores[0][0]==0:

    return individual_scores[0][1]

return individual_scores[0][1]
```

Running genetic optimizer

```
from ga_helloworld import *  
string_population=init_strings_population(2048)  
genetic_optimize(string_population, string_fitness,  
    mutate_string, string_crossover, 0.25,0.1,100)
```



mutation rate



elite percentage



max iterations

More useful problem: group travel

```
people = [('John', 'BOS'),  
          ('Mary', 'DAL'),  
          ('Laura', 'CAK'),  
          ('Abe', 'MIA'),  
          ('Greg', 'ORD'),  
          ('Lee', 'OMA')]  
  
# LaGuardia airport in New York  
destination='LGA'
```

The family members are from all over the country and wish to meet up in New York.

They will all arrive on the same day and leave on the same day, and they would like to share transportation to and from the airport.

There are about 9 flights per day to New York from any of the family members' locations, all leaving at different times.

The flights also vary in price and in duration.

Flight information

- The information about flights is in file ***schedule.txt***
- This file contains
origin, destination, departure time, arrival time,
and price

for a set of flights in a comma-separated format:

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

Adding flight info to the dictionary

```
flights={}
for line in file('schedule.txt'):
    origin,dest,depart,arrive,price
        =line.strip().split(',')
    flights.setdefault((origin,dest),[])
    # Add details to the list of possible flights
    flights[(origin,dest)].append(
        (depart,arrive,int(price)))
flights_index_range=[(0,9)]*(len(people)*2)
```

dictionary key


dictionary value: flight details variants
(list of size 10 for each key)

Representing solutions

- A very common representation is a list of numbers. In this case, each number can represent which flight a person chooses to take, where 0 is the first flight of the day, 1 is the second, and so on.
- Since each person needs an outbound flight and a return flight, the length of this list is twice the number of people.

For example, the list:

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



Represents a solution in which 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. Those are the positions in a list of flight details, we can interpret the flight details knowing this index and origin and destination of the flight

Fitness function design I

- The fitness function is the key to solving any problem using optimization, and it's usually the most difficult thing to determine.
- The goal of any optimization algorithm is to find a set of inputs—flights, in this case—that minimizes the cost function, so 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.

Fitness function design II

- Price

The total price of all the plane tickets, or possibly a weighted average that takes financial situations into account.

- Travel time

The total time that everyone has to 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 than when they rented it, or be forced to pay for a whole extra day.

Fitness function II

```
# Every person must wait at the airport until the latest person arrives.
# They also must arrive at the same time and wait for their flights on the way
back.

totalwait=0

for d in range(len(sol)/2):
    origin=people[d][1]
    outbound = flights[(origin,destination)][int(sol[2*d])]
    returnf = flights[(destination,origin)][int(sol[2*d+1])]
    totalwait+=latestarrival-getminutes(outbound[1])
    totalwait+=getminutes(returnf[0])-earliestdep

# Does this solution require an extra day of car rental? That'll be $50!
if latestarrival < earliestdep: totalprice+=5

return totalprice+totalwait
```

Execute GA for schedule optimization

```
execfile ("ga_schedule.py")
```

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

Tuning GA

- We could choose several variants of the algorithm, namely: breeding elite with the entire population, 2-points crossover etc.
- In order to have fine grained control over the computation, we have to adjust parameters such as population size, percentage of elite, mutation rate... Obviously these must be set empirically in order to fine tune the performance of the GA.

Other problems

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