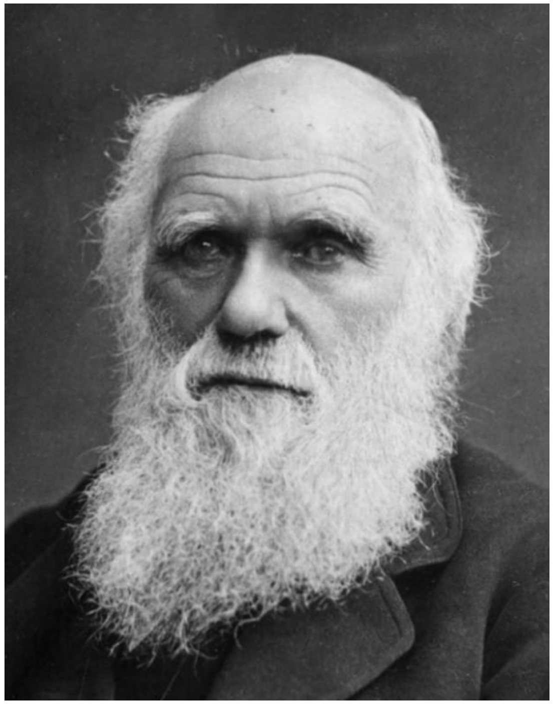
Machine Learning: Genetic Algorithms

Presented by: Derek Kane

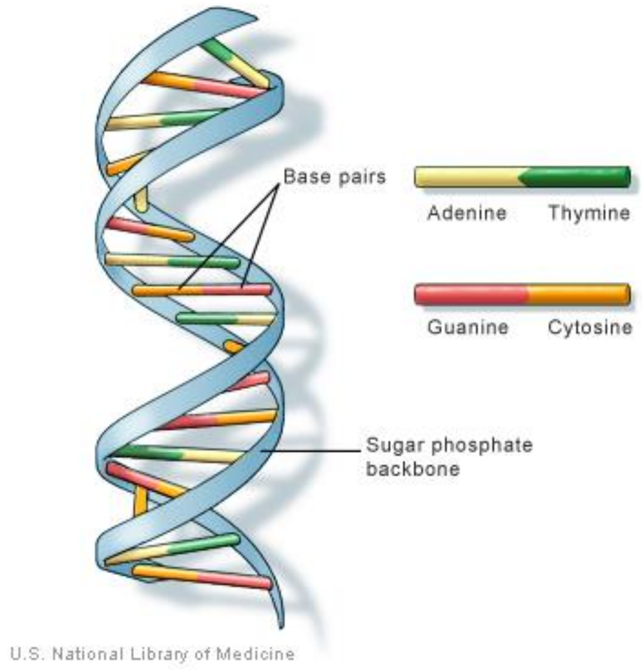
Overview of Topics

- What is a Genetic Algorithm?
- Biological Inspiration
- Evolution
- Algorithm Mechanics
- Practical Application Example
  - Knapsack Problem
  - Feature Selection
  - Traveling Salesman



Charles Darwin – "It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change."

What is a Genetic Algorithm?

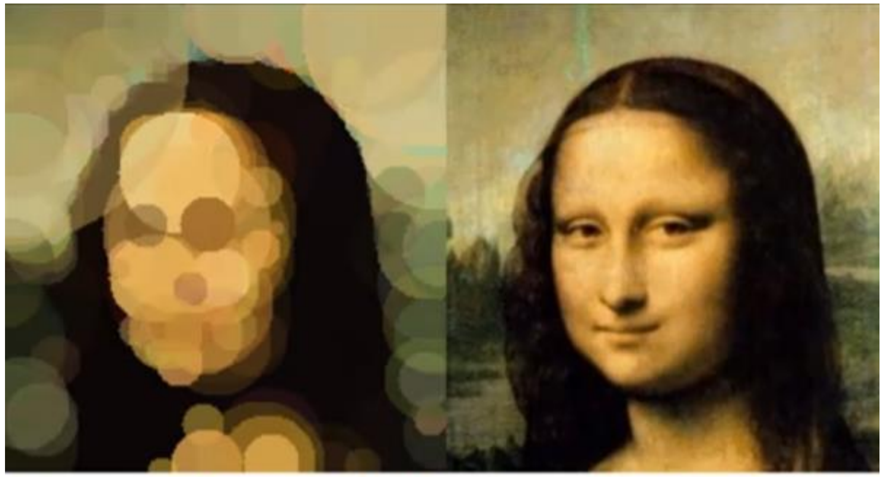


- A genetic algorithm is an adaptation procedure based on the mechanics of natural genetics and natural selection.
- Genetic Algorithm's have 2 essential components:
  - "Survival of the fittest"
  - Genetic Diversity
- Originally developed by John Holland (1975).
- The genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution.
- Uses concepts of "Natural Selection" and "Genetic Inheritance" (Darwin 1859).

What is a Genetic Algorithm?

Applications of Genetic Algorithms

- Optimization and Search Problems
- Scheduling and Timetabling
- Aerospace engineering
- Astronomy and astrophysics
- Chemistry
- Electrical engineering
- Financial markets
- Game playing
- Materials engineering
- Military and law enforcement
- Molecular biology
- Pattern recognition and data mining
- Robotics


Generation 162268 95.34% elapsed: 05:01:00.480702

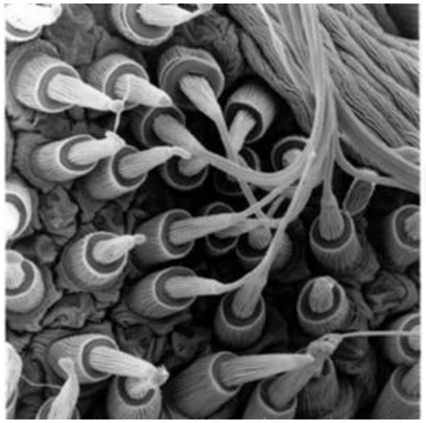
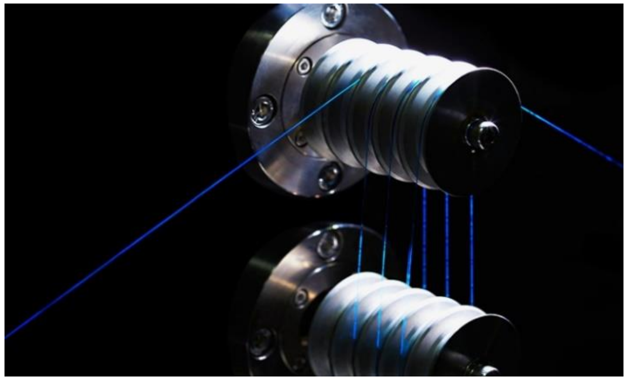
Nature's Diversity

- Nature is beautiful…

The Aye-Aye

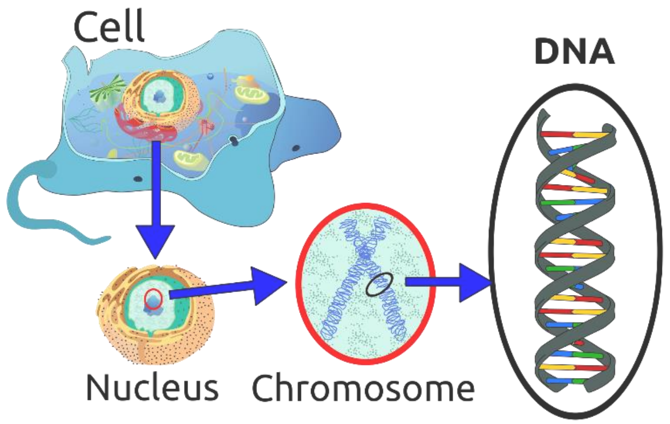
Applications of Nature

- What we can learn from nature?

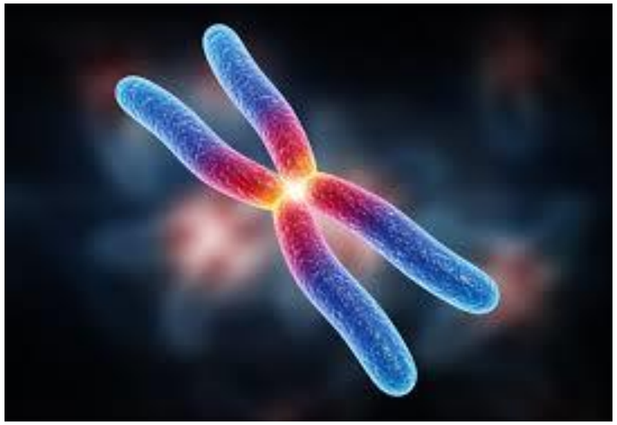



Evolution in the Real World

- To understand biological processes properly, we must first have an understanding of the cell.
- Human bodies are made up of trillions of cells.
- Each cell has a core structure (nucleus) that contains your chromosomes.
- Additionally, each of our 23 chromosomes are made up of tightly coiled strands of deoxyribonucleic acid (DNA).



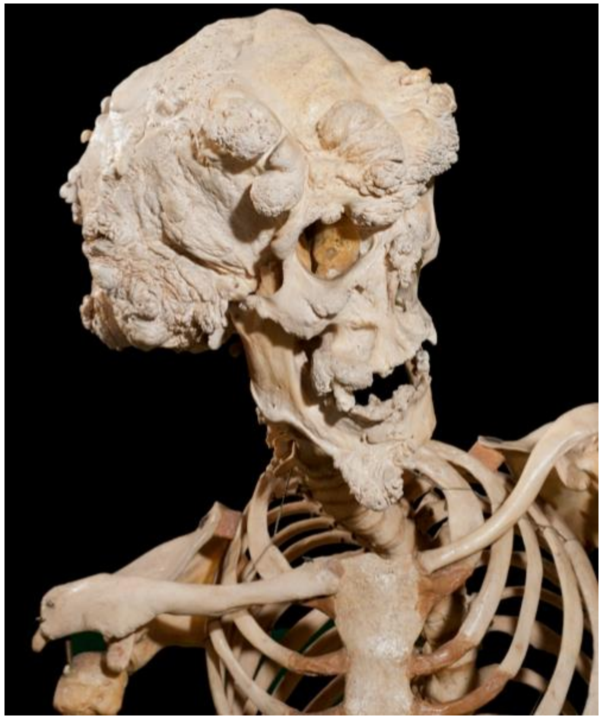
Evolution in the Real World



- Genes are segments of DNA that determine specific traits, such as eye or hair color.
- Humans have more than 20,000 genes. Each gene determines some aspect of the organism.
- A collection of genes is sometimes called a genotype.
- A collection of aspects (like eye characteristics) is sometimes called a phenotype.

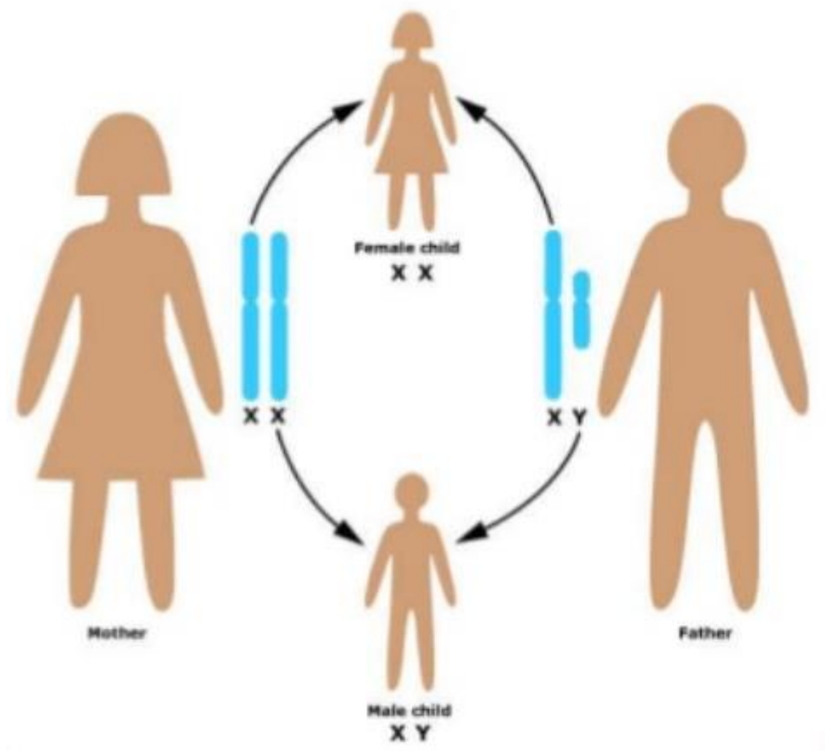
Evolution in the Real World

- A gene mutation is an alteration in your DNA.
- It can be inherited or acquired during your lifetime, as cells age or are exposed to certain chemicals.
- Mutations can also be triggered through errors within the DNA replication process.
- Some changes in your genes result in genetic disorders.



Joseph Merrick aka “The Elephant Man” is believed to have suffered from a genetic disorder called proteus syndrome.

Evolution in the Real World



- Reproduction involves recombination of genes from parents and then small amounts of mutation (errors) in copying.
- The fitness of an organism is how much it can reproduce before it dies.
- Here is an example of the passing of chromosomes within human reproduction.

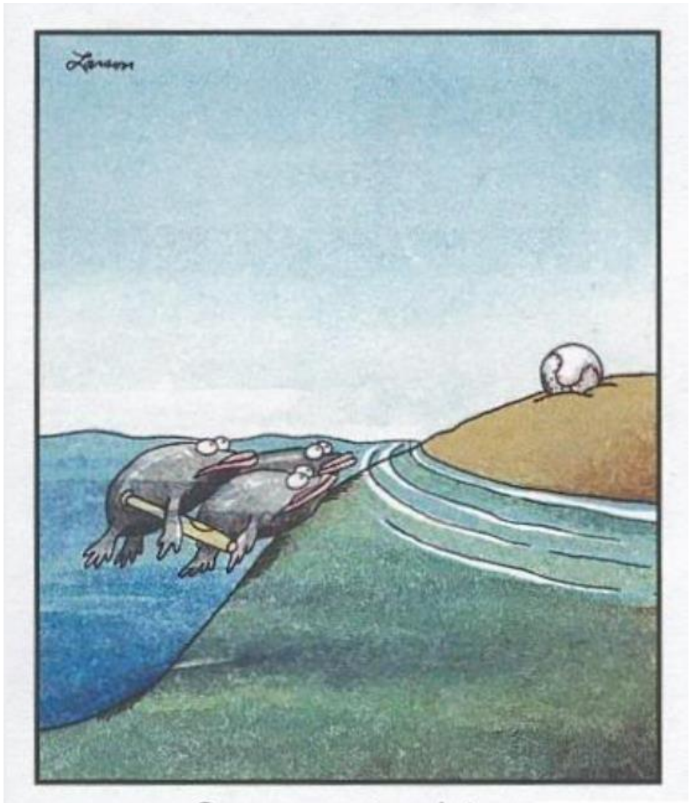
Evolution in the Real World

Natural Selection

Darwin's theory of evolution:

- Only the organisms best adapted to their environment tend to survive and transmit their genetic characteristics in increasing numbers to succeeding generations while those less adapted tend to be eliminated.
- A genetic algorithm maintains a population of candidate solutions for the problem at hand, and makes it evolve by iteratively applying a set of stochastic operators

Naturally Inspired Computing


Great moments in evolution

- The only intelligent systems on this planet are biological.
- Biological intelligences are designed by natural evolutionary processes.
- These intelligent organisms often work together in groups, swarms, or flocks.
- They don't appear to use logic, mathematics, complex planning, complicated modeling of their environment.
- They can achieve complex information processing and computational tasks that current artificial intelligences find very challenging indeed.

Naturally Inspired Computing

- Biological organisms cope with the demands of their environments.
- They uses solutions quite unlike the traditional human- engineered approaches to problem solving.
- They exchange information about what they've discovered in the places they have visited.
- Bio-inspired computing is a field devoted to tackling complex problems using computational methods modeled after design principles encountered in nature.

Classical Computing vs. Bio-Inspired

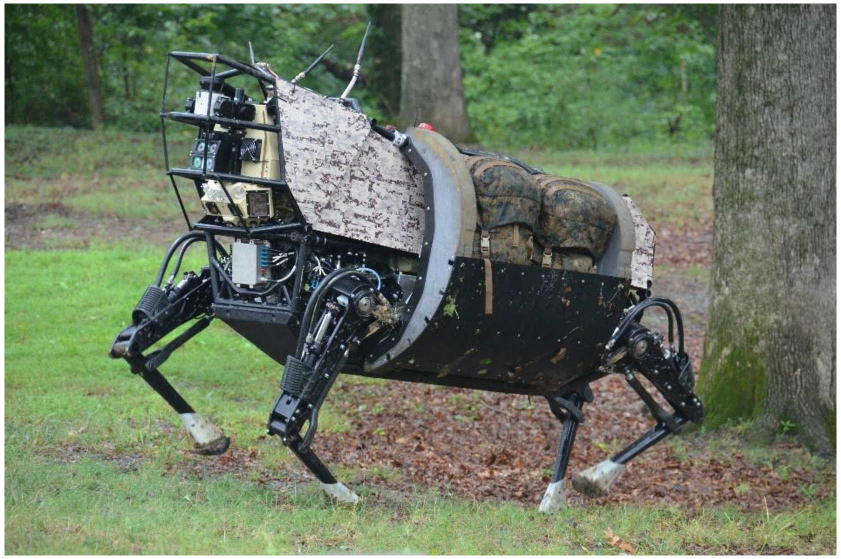
Classical computing's strengths:

- Number-crunching
- Thought-support (glorified pen-and-paper)
- Rule-based reasoning
- Constant repetition of well-defined actions.

Classical computing's weaknesses:

- Pattern recognition
- Robustness to damage
- Dealing with vague and incomplete information;
- Adapting and improving based on experience

Classical Computing vs. Bio-Inspired

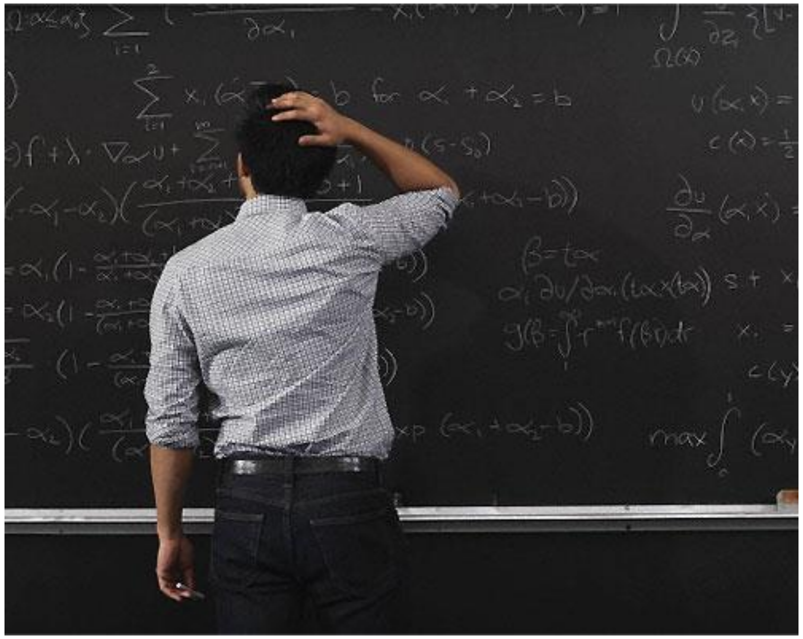

DARPA - Legged Squat Support System (LS3)

- Bio-inspired computing takes a more evolutionary approach to learning.
- In traditional AI, the intelligence is often programmed from above. The Programmer creates the program and imbues it with its intelligence.
- Bio-inspired computing, on the other hand, takes a more bottom-up, decentralized approach.
- Bio-inspired computing often involve the method of specifying a set of simple rules, a set of simple organisms which adhere to those rules.

Problem Solving

Suppose you have a problem.

- ...And you don't know how to solve it.
- What can you do?
- Can you use a computer to somehow find a solution?
- This would be nice! Can it be done?



Problem Solving

Brute-Force Solution:

- A “blind generate and test” algorithm.
- Repeat
- Generate a random possible solution
- Test the solution and see how the solution performed.
- Stop once the solution is good enough.

Problem Solving

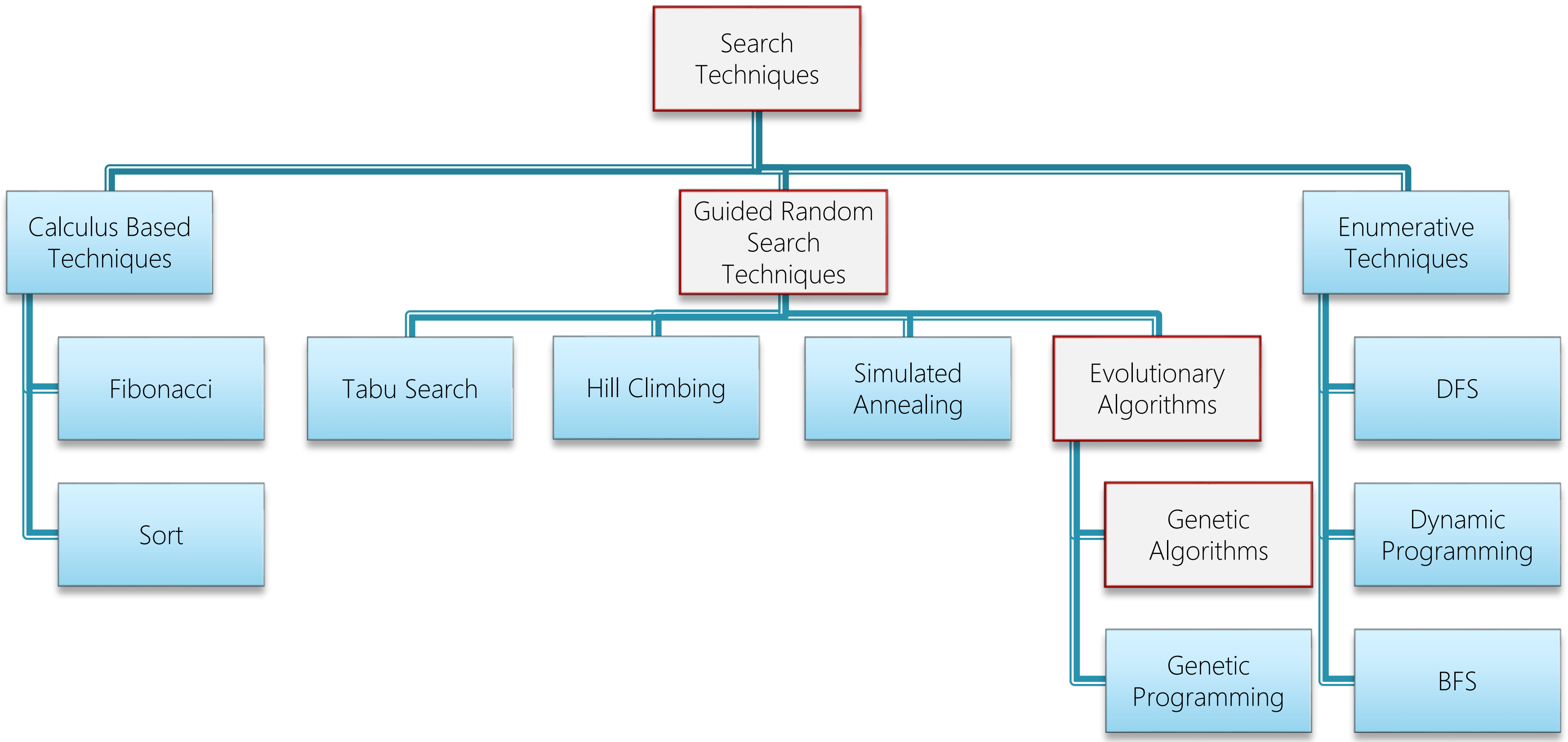
Can we use this Brute-Force idea?

- Sometimes - YES
  - if there are only a few possible solutions
  - and you have enough time
  - then such a method could be used
- For most problems - NO
  - many possible solutions
  - with no time to try them all
  - Therefore, this method cannot be used

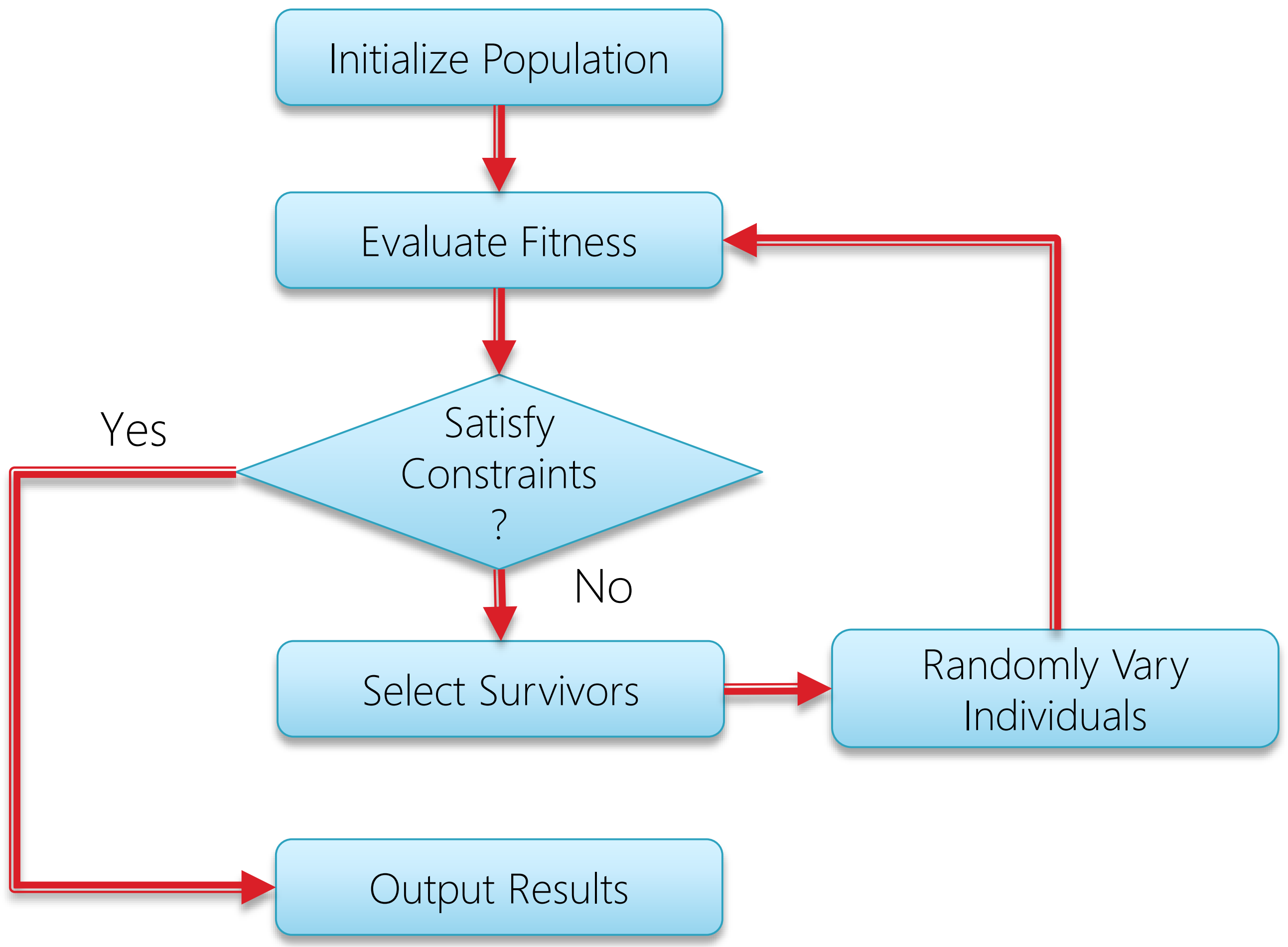
# Data Points	# of Solutions
5	12
6	60
7	360
8	2520
9	20160
10	181440

Key Point: The total number of solutions for 25 data points is 310,224,200,866,619,719,680,000.

Problem Solving



Genetic Algorithms



Genetic Algorithms

How do you encode a solution?

- This depends on the problem we are trying to solve with genetic algorithms.
- Genetic algorithm's often encode solutions as fixed length "bitstrings" (e.g. 101110, 111111, 000101) which can also be thought of as chromosomes.
- Each bit represents some aspect of the proposed solution to the problem.
- For Genetic Algorithm's to work, we need to be able to "test" any string and get a "score" indicating how "good" that solution is.

Genetic Algorithms

- The set of all possible solutions [0 to 1000] is called the *search space* or state space.
- In this example, it's just one number but it could be many numbers.
- Often genetic algorithms code numbers in binary producing a bitstring representing a solution.
- We choose 1,0 bits which is enough to represent 0 to 1000

Binary Representation

$$\underbrace{1000}_{\text{Param 1}}\ \underbrace{1011}_{\text{Param 2}}\ \underbrace{0110}_{\text{Param 3}}\ \underbrace{1111}_{\text{Param 4}}$$

Encoding 4 Parameters:

Parameter 1 = 1000 = 4

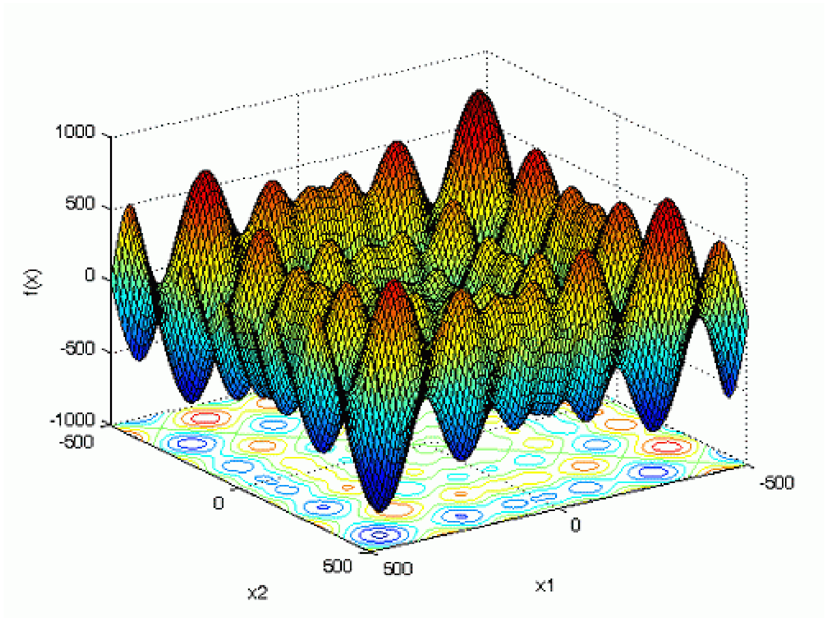
Parameter 2 = 1011 = 7

Etc...

Genetic Algorithms

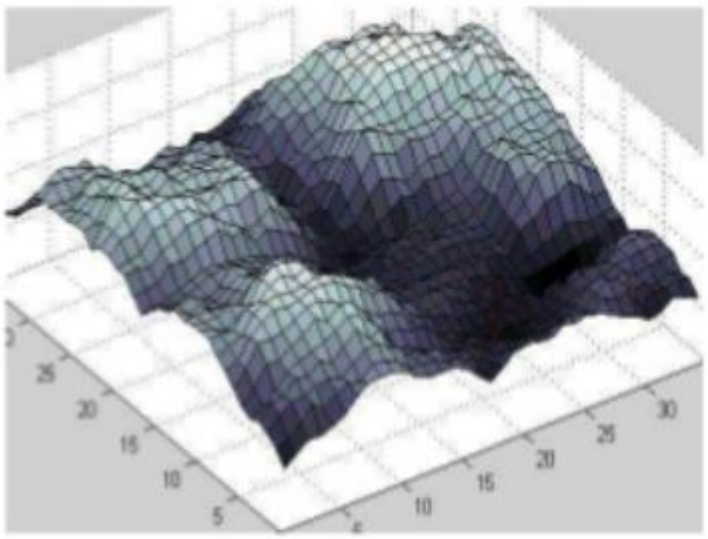
Search Space

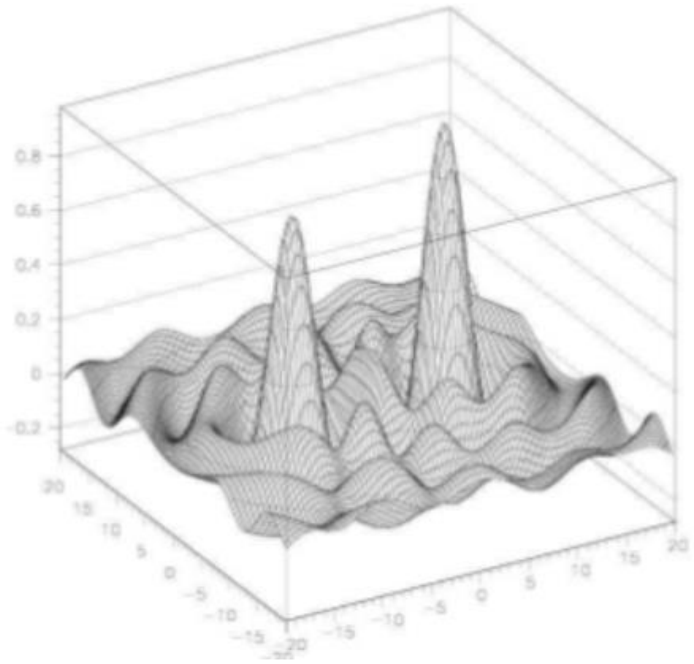
- For a simple function f(x) the search space is one dimensional.
- But by encoding several values into the chromosome many dimensions can be searched e.g. two dimensions f(x,y).
- The search space can be visualized as a surface or fitness landscape in which fitness dictates height.
- Each possible genotype is a point in the space.
- A genetic algorithm tries to move the points to better places (higher fitness) in the space.

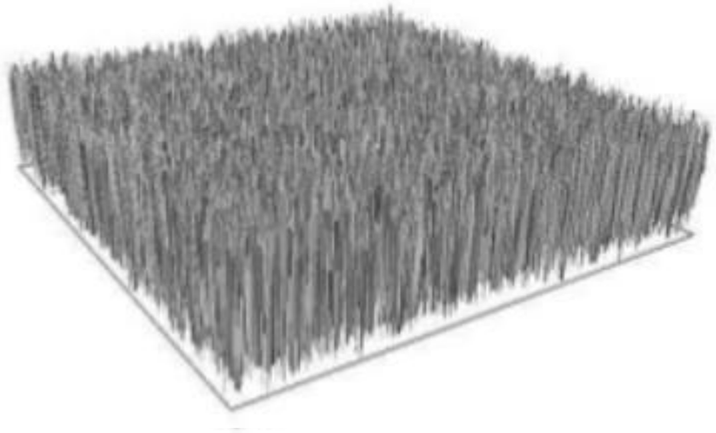


Genetic Algorithms

- Various fitness landscapes







Genetic Algorithms

Implicit fitness functions

- Most GA's use explicit and static fitness function.

- Some genetic algorithm's (such as in Artificial Life or Evolutionary Robotics) use dynamic and implicit fitness functions - like "how many obstacles did I avoid".

$$\frac{Individual's\ fitness}{Average\ fitness\ of\ population}$$

Genetic Algorithms

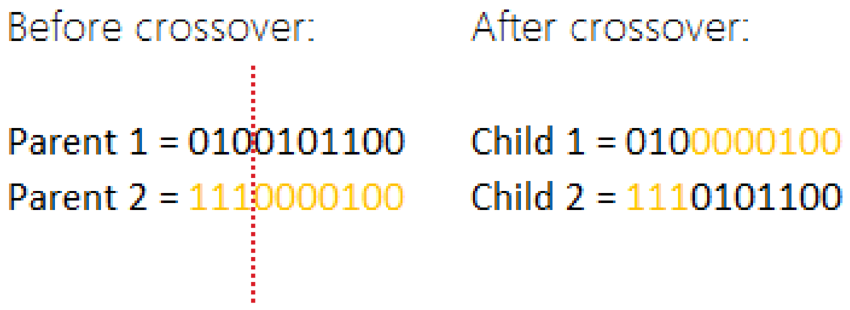
Selecting Parents for breeding

- Many schemes are possible so long as better scoring chromosomes are more likely selected.
- Score is often termed the "fitness"

"Roulette Wheel" selection can be used:

- Add up the fitness's of all chromosomes
- Generate a random number R in that range
- Select the first chromosome in the population that - when all previous fitness's are added - gives you, at a minimum, the value R

Genetic Algorithms



Before crossover:

Parent 1 = 0100101100
Parent 2 = 1110000100

After crossover:

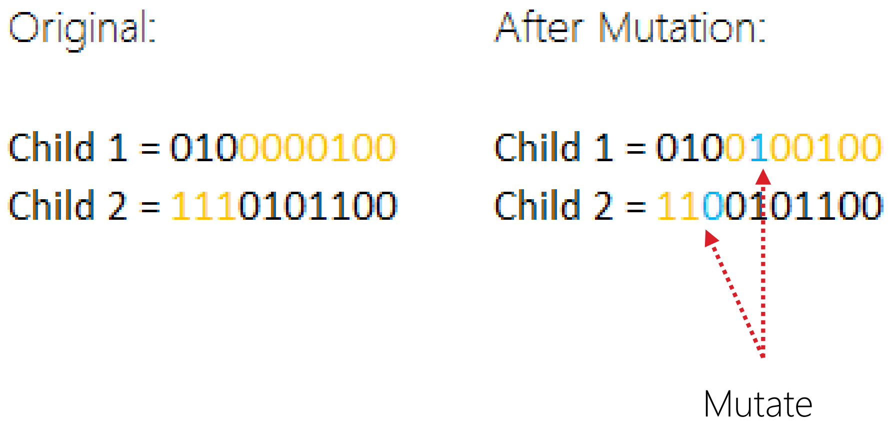
Child 1 = 0100000100
Child 2 = 1110101100

Crossover single point - random

- The crossover point is a single point identified at random between two chromosomes.
- A crossover rate is pre determined (using a high probability number like 0.8 to 0.95) and then the crossover is applied to the parents.
- The idea is that crossover preserves "good bits" from different parents, combining them to produce better solutions.
- A good encoding scheme would therefore try to preserve "good bits" during crossover and mutation.

Genetic Algorithms

- With some small probability (the mutation rate), we flip each bit in the offspring.
- Typical values for the mutation rate are very small and are usually values between 0.1 and 0.001.
- Causes movement in the search space (local or global).
- Restores lost information to the population.



Original:

Child 1 = 0100000100

Child 2 = 1110101100

After Mutation:

Child 1 = 0100100100

Child 2 = 1100101100

Anatomy of Genetic Algorithm

There are many variants of GA

- Different kinds of selection (not roulette)
  - Tournament
  - Elitism, etc.

- Different recombination procedures
  - Multi-point crossover
  - 3 way crossover, etc.

- Different kinds of encoding other than bitstring.
  - Integer values
  - Ordered set of symbols

- Different kinds of mutation

Genetic Algorithm Considerations

GA implementation considerations:

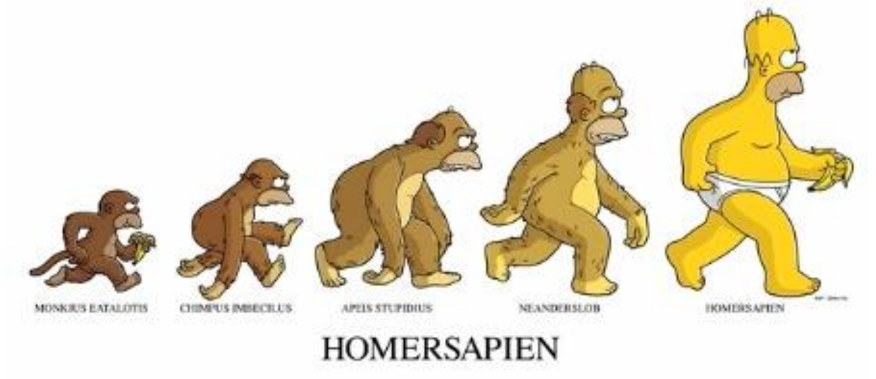
- Representation
- population size, mutation rate, …
- selection, deletion policies
- crossover, mutation operators
- Termination Criteria
- Performance, scalability
- The Solution is only as good as the evaluation function (often hardest part)

Genetic Algorithms

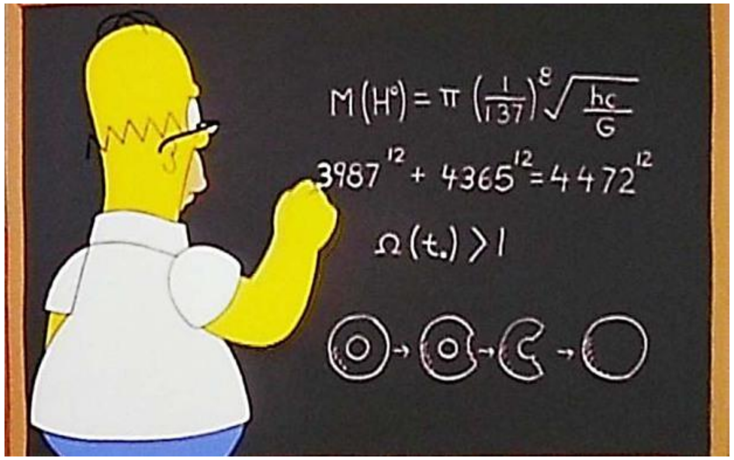
Lets go through a simple example to showcase the steps.

Basic Genetic Algorithm Process:

- Produce an initial population of individuals
- Evaluate the fitness of all individuals
- while termination condition not met do
  - select fitter individuals for reproduction
  - recombine between individuals
  - mutate individuals
  - evaluate the fitness of the modified individuals
  - generate a new population
- End while



Genetic Algorithm Example


$M(H^0) = \pi \left(\frac{1}{137}\right)^8 \sqrt{\frac{hc}{G}}$
$3987^{12} + 4365^{12} = 4472^{12}$
$\Omega(t_0) > 1$

The MAXONE problem:

- Suppose we want to maximize the number of ones in a string of L binary digits.
- It may seem trivial 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 L yes/no difficult questions.

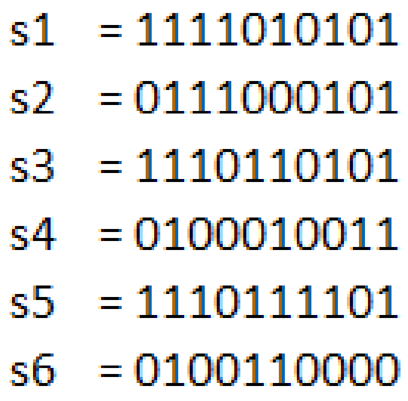
Encoding

- An individual is encoded (naturally) as a string of L binary digits
- Let's say L = 10. Then, 1 = 0000000001 (10 bits)

Genetic Algorithm Example

- We start with a population of n random strings. Suppose that L = 10 and n = 6

- We toss a fair coin 60 times and get the following initial population:



s1 = 1111010101
s2 = 0111000101
s3 = 1110110101
s4 = 0100010011
s5 = 1110111101
s6 = 0100110000

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
End while
```

Genetic Algorithm Example

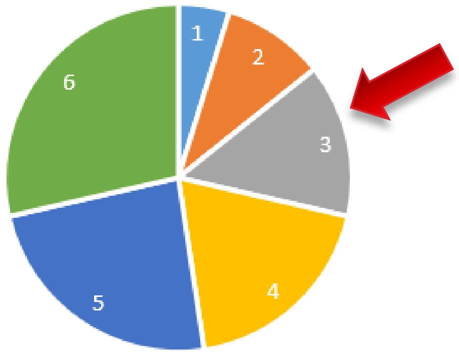
- The fitness function: f (x)
- We toss a fair coin 60 times and get the following initial population:

s1	= 1111010101	f (s1) = 7
s2	= 0111000101	f (s2) = 5
s3	= 1110110101	f (s3) = 7
s4	= 0100010011	f (s4) = 4
s5	= 1110111101	f (s5) = 8
s6	= 0100110000	f (s6) = 3
		= 34

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
End while
```

Genetic Algorithm Example

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



Area is Proportional to Fitness Value

Individual *i* will have a probability to be chosen

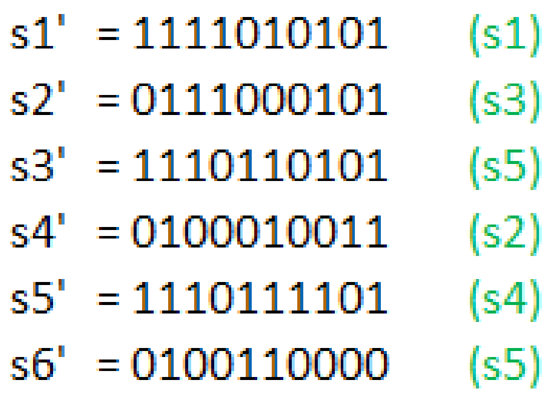
$$p = \frac{f(i)}{\sum_i f(i)}$$

- 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).

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
End while
```

Genetic Algorithm Example

- Suppose that, after performing selection, we get the following population:



s1'	= 1111010101	(s1)
s2'	= 0111000101	(s3)
s3'	= 1110110101	(s5)
s4'	= 0100010011	(s2)
s5'	= 1110111101	(s4)
s6'	= 0100110000	(s5)

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
End while
```

Genetic Algorithm Example

- 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 (s1`, s2`) and (s5`, s6`).
- For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second .

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
End while
```

Before crossover:

s1' = 1111010101   s5' = 0100010011
s2' = 1110110101   s6' = 1110111101

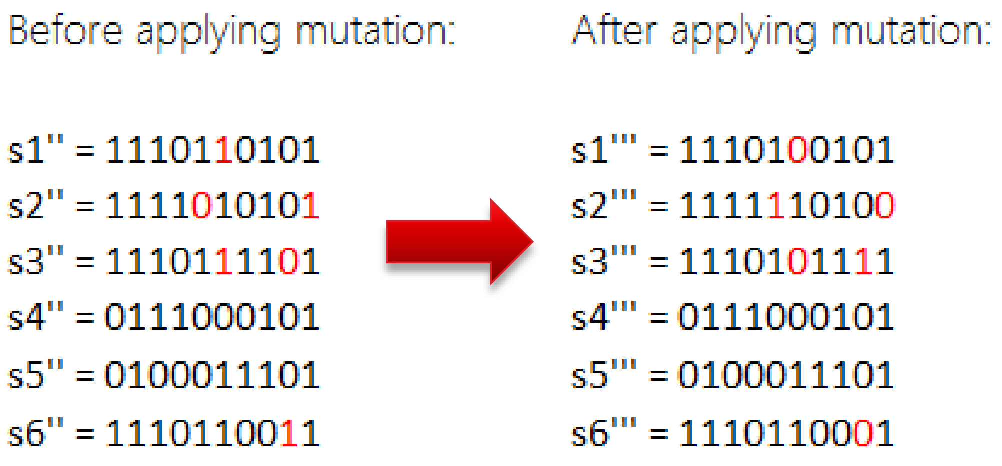
After crossover:

s1'' = 1110110101   s5'' = 0100011101
s2'' = 1111010101   s6'' = 1110110011

Genetic Algorithm Example

- 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)
- Causes movement in the search space (local or global)
- Restores lost information to the population

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
End while
```



Before applying mutation:

s1'' = 1110110101
s2'' = 1111010101
s3'' = 1110111101
s4'' = 0111000101
s5'' = 0100011101
s6'' = 1110110011

After applying mutation:

s1''' = 1110100101
s2''' = 1111110100
s3''' = 1110101111
s4''' = 0111000101
s5''' = 0100011101
s6''' = 1110110001

Genetic Algorithm Example

- After applying mutation:

s1''' = 1110100101	f (s1''') = 6
s2''' = 1111110100	f (s2''') = 7
s3''' = 1110101111	f (s3''') = 8
s4''' = 0111000101	f (s4''') = 5
s5''' = 0100011101	f (s5''') = 5
s6''' = 1110110001	f (s6''') = 6
	= 37

```
Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met do
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
End while
```

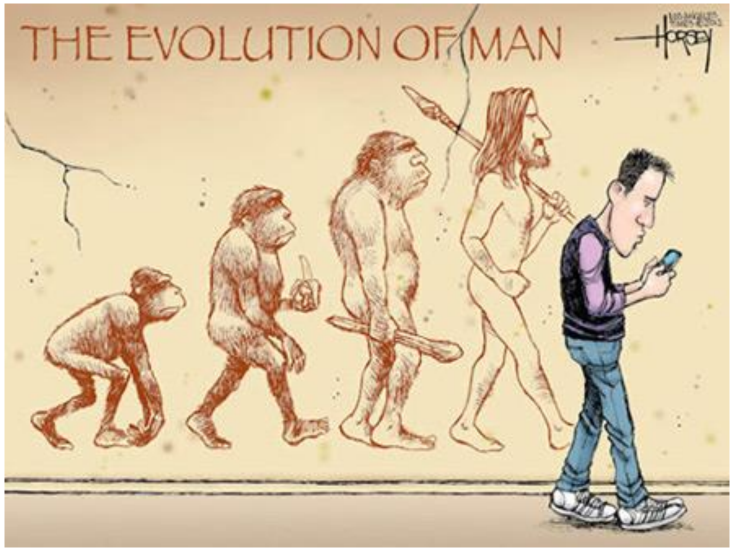
- In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%
- At this point, we go through the same process all over again, until a stopping criterion is met

Genetic Algorithms

Advantages:

- Concepts are easy to understand
- Inspired from nature
- Has many areas of applications
- GA is powerful
- Genetic Algorithms are intrinsically parallel
- Always an answer; answer gets better with time.
- Inherently parallel; easily distributed.
- Less time required for some special applications.
- Chances of getting the optimal solution are greater.

Genetic Algorithms

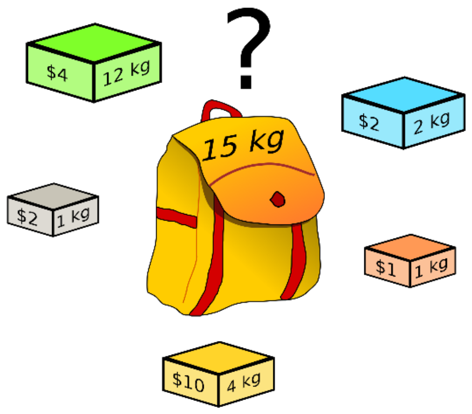


Limitations

- The population considered for the evolution should be moderate or suitable one for the problem (normally 20-30 or 50- 100)
- Crossover rate should be 80%-95%
- Mutation rate should be low i.e. 0.5%-1% assumed as best
- The method of selection should be appropriate for the problem.
- Writing of fitness function must be accurate.

Practical Example – Knapsack Problem

The Knapsack Problem



- The knapsack problem can be thought of as the following:
- "You are going to spend a month in the wilderness. You're taking a backpack with you, however, the maximum weight it can carry is 20 kilograms. You have a number of survival items available, each with its own number of "survival points". You're objective is to maximize the number of survival points."
- This type of problem is called a constrained optimization problem because we are limited to carrying a maximum weight of 20 kilograms.
- Our goal is to devise:
  - A genetic algorithm to approach solving the knapsack problem.

Understanding the data

- In order to better prepare the analysis, we must first understand the data we are working with.

item	survivalpoints	weight
pocketknife	10	1
beans	20	5
potatoes	15	10
unions	2	1
sleeping bag	30	7
rope	10	5
compass	30	1

Each row will be encoded into a chromosome bitstring. A single row represents a single value in the bitstring.

- There are 7 row entries in our chromosome and a 0 will represent "do not include" and 1 represents "include".

- Therefore, an example of a chromosome would be "1001011"

The Knapsack Problem

- Here is the code to create the dataset:

```
# Create the table for the knapsack problem.

dataset <- data.frame(item = c("pocketknife", "beans", "potatoes",
                               "unions", "sleeping bag", "rope", "compass"),
                      survivalpoints = c(10, 20, 15, 2, 30, 10, 30),
                      weight = c(1, 5, 10, 1, 7, 5, 1))

# Create a variable to represent the maximum weightlimit.

weightlimit <- 20
```

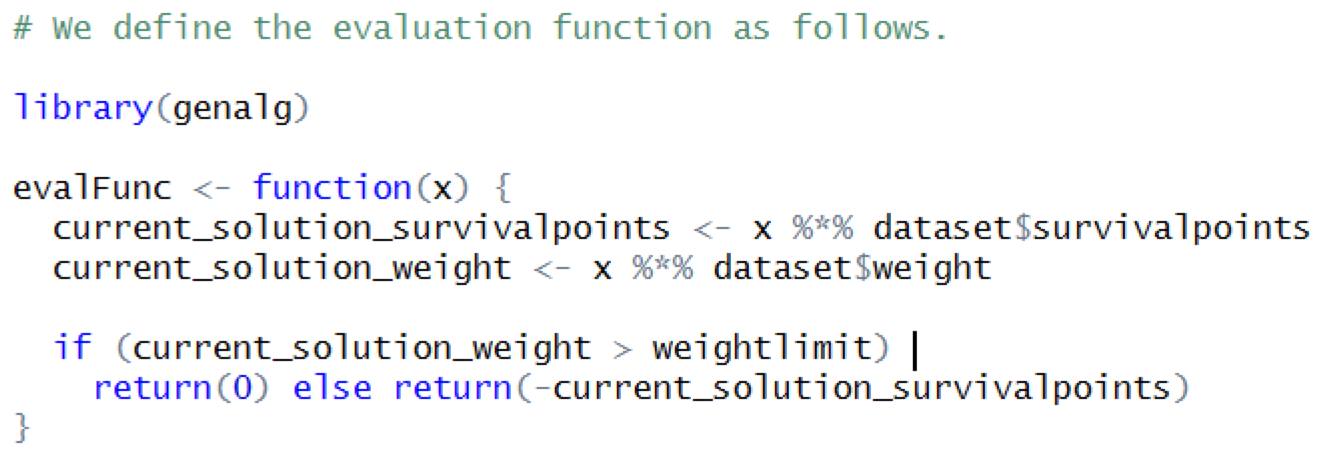
- This will show us how to create a chromosome to include the first, fourth, and fifth item.

```
# Given the example gene configuration we would take the following items;

chromosome = c(1, 0, 0, 1, 1, 0, 0)
dataset[chromosome == 1, ]

# We can check to what amount of surivival points this configuration sums up.

cat(chromosome %*% dataset$survivalpoints)
```

The Knapsack Problem

- The genalg algorithm tries to optimize towards the minimum value. Therefore, the value is calculated as above and multiplied with -1.

- A configuration which leads to exceeding the weight constraint returns a value of 0 (a higher value can also be given).



```r
# We define the evaluation function as follows.

library(genalg)

evalFunc <- function(x) {
  current_solution_survivalpoints <- x %*% dataset$survivalpoints
  current_solution_weight <- x %*% dataset$weight

  if (current_solution_weight > weightlimit)
    return(0) else return(-current_solution_survivalpoints)
}
```

The Knapsack Problem

- Next, we choose the number of iterations, design and run the model.

```
GAmodel <- rbga.bin(size = 7, popSize = 200, iters = 100, mutationChance = 0.01,
                    elitism = T, evalFunc = evalFunc)
```

- Notice that within the settings of genalg, the type is a binary chromosome by default:

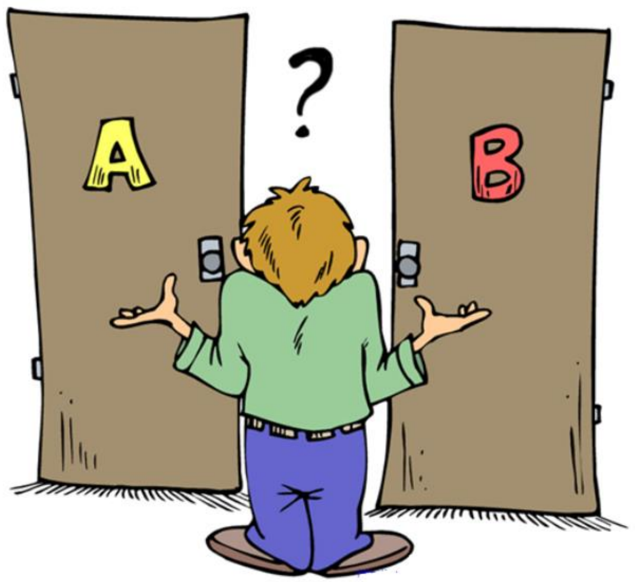
```
GA Settings
  Type                  = binary chromosome
  Population size       = 200
  Number of Generations = 100
  Elitism               = TRUE
  Mutation Chance       = 0.01

Search Domain
  Var 1 = [,]
  Var 0 = [,]

GA Results
  Best Solution : 1 1 0 1 1 1 1
```

Optimal Knapsack Configuration

item	survivalpoints	weight
pocketknife	10	1
beans	20	5
unions	2	1
sleeping bag	30	7
rope	10	5
compass	30	1

Practical Example – Feature Selection & OLS

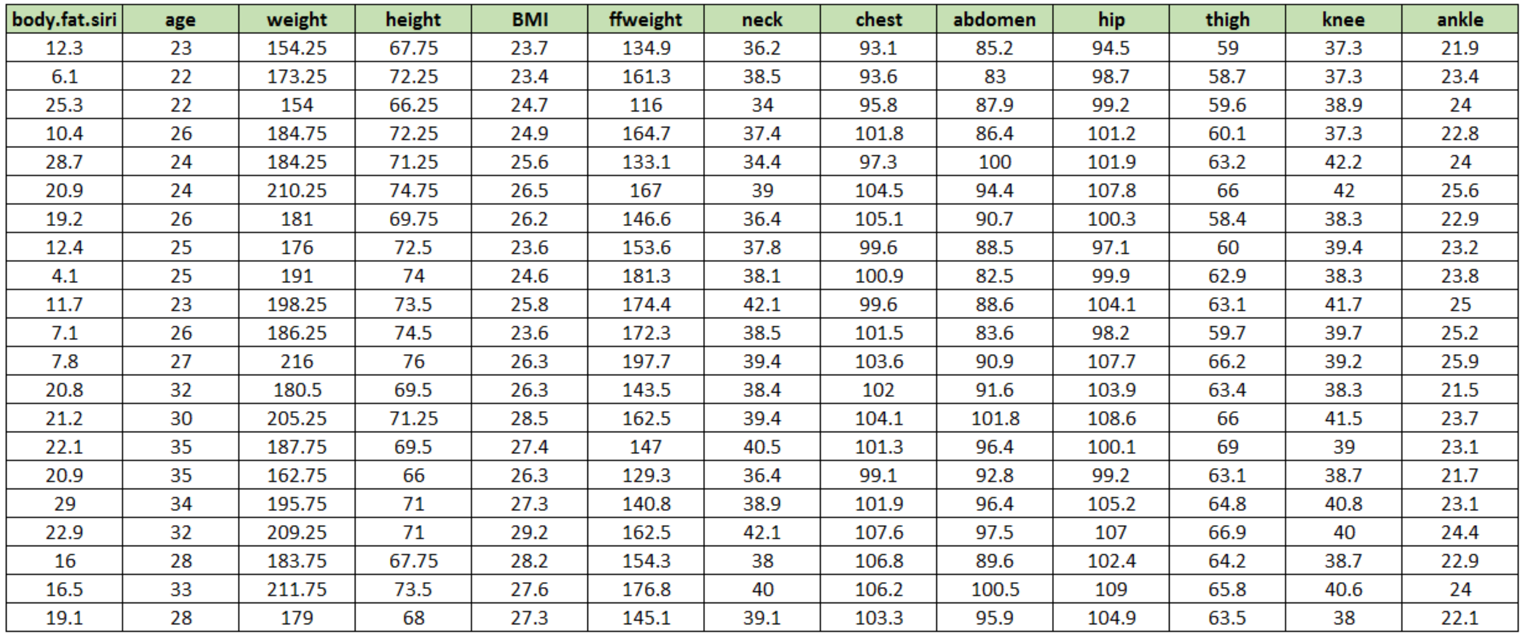
Feature Selection



- When we approach machine learning tasks, we are often confronted with many variables within a particular dataset which can be used to build our predictive models.
- In previous lectures we discussed employing some techniques that draw from AIC and BIC to reduce the candidate variable pool. These techniques include the forward, backward, and stepwise techniques.
- Now lets explore using genetic algorithms for this feature selection.
- Our goal is to devise:
  - A genetic algorithm to approach feature selection.
  - We will also estimate the coefficients for an OLS regression model using Genetic Algorithms

Understanding the data

- Here is the “fat” dataset that we are working with from the package UsingR.



body.fat.siri	age	weight	height	BMI	ffweight	neck	chest	abdomen	hip	thigh	knee	ankle
12.3	23	154.25	67.75	23.7	134.9	36.2	93.1	85.2	94.5	59	37.3	21.9
6.1	22	173.25	72.25	23.4	161.3	38.5	93.6	83	98.7	58.7	37.3	23.4
25.3	22	154	66.25	24.7	116	34	95.8	87.9	99.2	59.6	38.9	24
10.4	26	184.75	72.25	24.9	164.7	37.4	101.8	86.4	101.2	60.1	37.3	22.8
28.7	24	184.25	71.25	25.6	133.1	34.4	97.3	100	101.9	63.2	42.2	24
20.9	24	210.25	74.75	26.5	167	39	104.5	94.4	107.8	66	42	25.6
19.2	26	181	69.75	26.2	146.6	36.4	105.1	90.7	100.3	58.4	38.3	22.9
12.4	25	176	72.5	23.6	153.6	37.8	99.6	88.5	97.1	60	39.4	23.2
4.1	25	191	74	24.6	181.3	38.1	100.9	82.5	99.9	62.9	38.3	23.8
11.7	23	198.25	73.5	25.8	174.4	42.1	99.6	88.6	104.1	63.1	41.7	25
7.1	26	186.25	74.5	23.6	172.3	38.5	101.5	83.6	98.2	59.7	39.7	25.2
7.8	27	216	76	26.3	197.7	39.4	103.6	90.9	107.7	66.2	39.2	25.9
20.8	32	180.5	69.5	26.3	143.5	38.4	102	91.6	103.9	63.4	38.3	21.5
21.2	30	205.25	71.25	28.5	162.5	39.4	104.1	101.8	108.6	66	41.5	23.7
22.1	35	187.75	69.5	27.4	147	40.5	101.3	96.4	100.1	69	39	23.1
20.9	35	162.75	66	26.3	129.3	36.4	99.1	92.8	99.2	63.1	38.7	21.7
29	34	195.75	71	27.3	140.8	38.9	101.9	96.4	105.2	64.8	40.8	23.1
22.9	32	209.25	71	29.2	162.5	42.1	107.6	97.5	107	66.9	40	24.4
16	28	183.75	67.75	28.2	154.3	38	106.8	89.6	102.4	64.2	38.7	22.9
16.5	33	211.75	73.5	27.6	176.8	40	106.2	100.5	109	65.8	40.6	24
19.1	28	179	68	27.3	145.1	39.1	103.3	95.9	104.9	63.5	38	22.1

- Our goal is to create a linear regression model using “body.fat.siri” as the dependent variable and some ideal subset of variables as the independent variables.

Feature Selection

- Lets first create a linear regression model

Regression Formula
lm(formula = body.fat.siri ~ age + weight + height + neck + chest + abdomen + hip + thigh + knee + ankle + bicep + forearm + wrist, data = mydata)

Residuals				
Min	1Q	Median	3Q	Max
-11.1687	-2.8639	-0.1014	3.2085	10.0068

Residual standard error:	4.305 on 238 degrees of freedom
Multiple R-Squared:	0.749
Adjusted R-Squared:	0.7353
F-Statistic:	54.65 on 13 and 238 DF
p-value:	< 2.2e-16

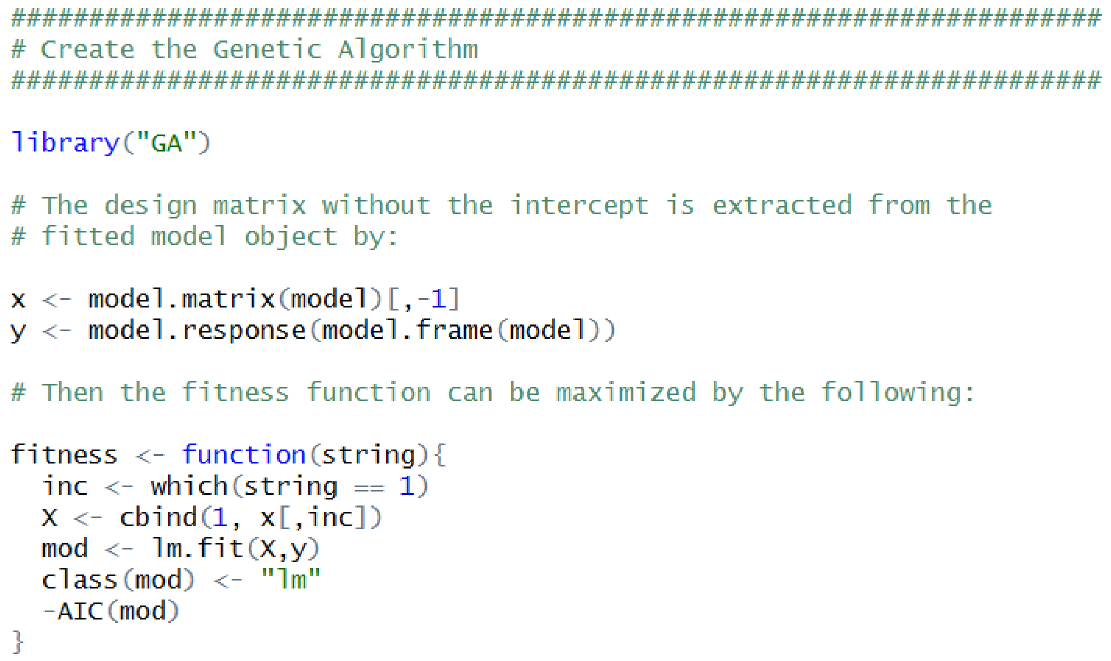
Coefficients:

Parameter	Estimate	Std. Error	t value	Pr>|t|
(Intercept)	-18.1885	17.34857	-1.048	2.96E-01
age	0.06208	0.03235	1.919	5.62E-02
weight	-0.08844	0.05353	-1.652	0.09978
height	-0.06959	0.09601	-0.725	0.46925
neck	-0.4706	0.23247	-2.024	0.04405
chest	-0.02386	0.09915	-0.241	0.81
abdomen	0.95477	0.08645	11.044	< 2.00E-16
hip	-0.20754	0.14591	-1.422	0.15622
thigh	0.2361	0.14436	1.636	0.10326
knee	0.01528	0.24198	0.063	0.9497
ankle	0.174	0.22147	0.786	0.43285
bicep	0.1816	0.17113	1.061	0.28966
forearm	0.45202	0.19913	2.27	0.0241
wrist	-1.62064	0.53495	-3.03	0.00272

- **Note:** The adjusted R squared is 0.7353 and there are a number of statistically insignificant variables at the 0.05 level.

Feature Selection

- We then develop some additional R code to prepare the model to run using the following:



```
######################################################################
# Create the Genetic Algorithm
######################################################################

library("GA")

# The design matrix without the intercept is extracted from the
# fitted model object by:

x <- model.matrix(model)[,-1]
y <- model.response(model.frame(model))

# Then the fitness function can be maximized by the following:

fitness <- function(string){
  inc <- which(string == 1)
  X <- cbind(1, x[,inc])
  mod <- lm.fit(X,y)
  class(mod) <- "lm"
  -AIC(mod)
}
```

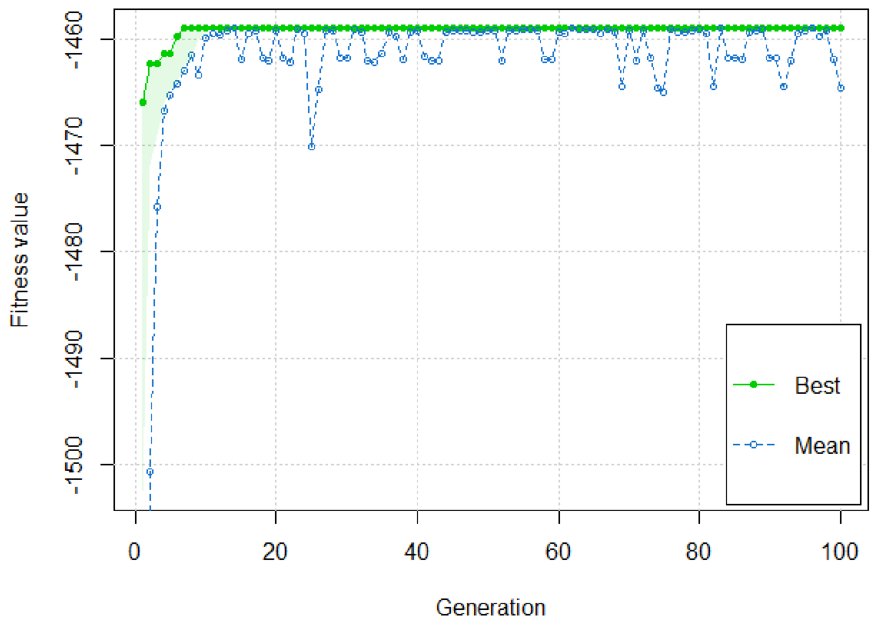
- This fitness function estimates the regression model using predictors identified by a 1 in the corresponding position of the string.
- Each column (variable) is converted into a binary string with 1 = include.

Feature Selection

- Now we run the genetic algorithm.

```
GA <- ga("binary", fitness = fitness, nBits = ncol(x), names= colnames(x), monitor=plot)
```

```
+-----------------------------------+
|         Genetic Algorithm         |
+-----------------------------------+

GA settings:
Type                  =  binary
Population size       =  50
Number of generations =  100
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 100
Fitness function value = -1458.996
```



```
Solution              =
    age weight height neck chest abdomen hip thigh knee ankle bicep forearm wrist
     1     1      0    1     0       1   1     1    0     0     0       1     1
```

Feature Selection

- This code will take the results of the genetic algorithm and pass them as a parameter:

```
model2 <- lm(body.fat.siri~.,
             data=data.frame(body.fat.siri = y, x[,GA@solution == 1]))
```

Residuals				
Min	1Q	Median	3Q	Max
-10.9757	-2.9937	-0.1644	2.9766	10.2244

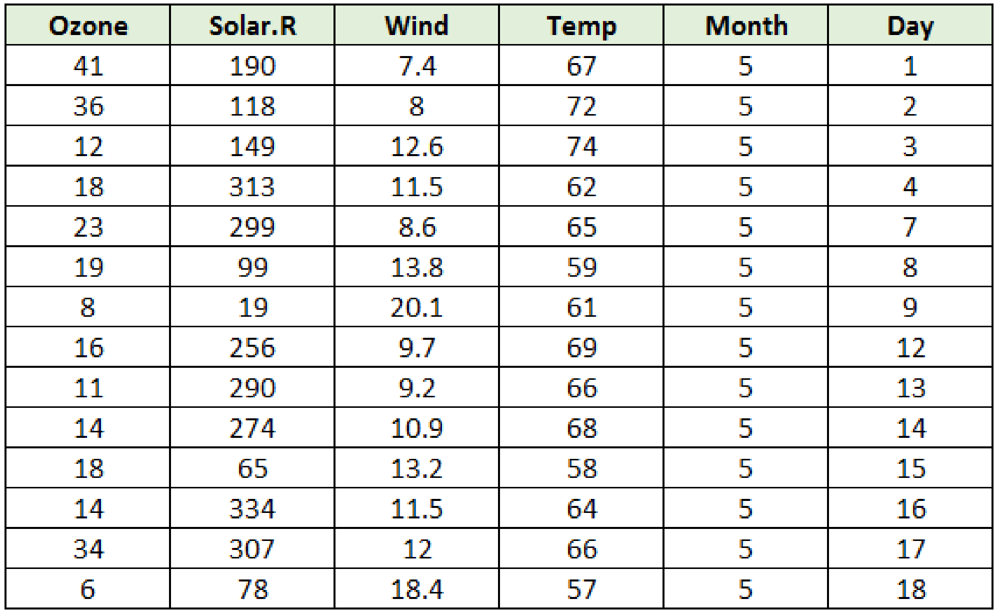
Residual standard error:	4.282 on 243 degrees of freedom
Multiple R-Squared:	0.7466
Adjusted R-Squared:	0.7382
F-Statistic:	89.47 on 8 and 243 DF
p-value:	< 2.2e-16

Coefficients:				
Parameter	**Estimate**	**Std. Error**	**t value**	**Pr>|t|**
(Intercept)	-22.65637	11.71385	-1.934	5.43E-02
age	0.06578	0.03078	2.137	3.36E-02
weight	-0.08985	0.03991	-2.252	0.02524
neck	-0.46656	0.22462	-2.077	0.03884
abdomen	0.94482	0.07193	13.134	<2E-16
hip	-0.19543	0.13847	-1.411	0.1594
thigh	0.30239	0.12904	2.343	0.01992
forearm	0.51572	0.18631	2.768	0.00607
wrist	-1.53665	0.50939	-3.017	0.00283

- The Adjusted R squared is slightly better at 0.7382.
- Now imagine combining genetic algorithm's with artificial neural networks…

OLS Parameter Estimate with GA

- Here is an example of an interesting approach to using a genetic algorithm to approximate the β coefficients. Lets use the following airquality dataset and the GA package.

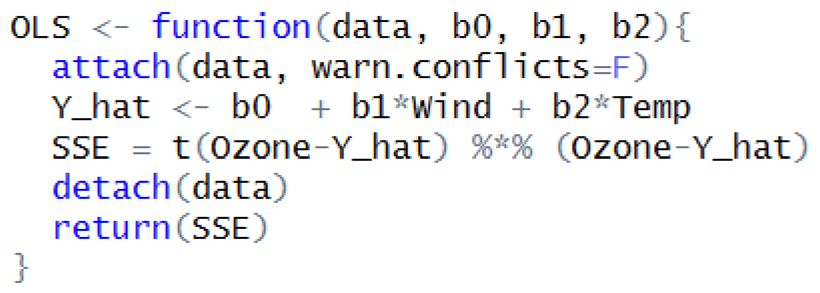


Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
23	299	8.6	65	5	7
19	99	13.8	59	5	8
8	19	20.1	61	5	9
16	256	9.7	69	5	12
11	290	9.2	66	5	13
14	274	10.9	68	5	14
18	65	13.2	58	5	15
14	334	11.5	64	5	16
34	307	12	66	5	17
6	78	18.4	57	5	18

- We are going to build a linear regression model with ozone as the dependent variable and wind & temp as the independent variables.

OLS Parameter Estimate with GA

- This is our fitness function for our genetic algorithm.



```
OLS <- function(data, b0, b1, b2){
  attach(data, warn.conflicts=F)
  Y_hat <- b0  + b1*Wind + b2*Temp
  SSE = t(Ozone-Y_hat) %*% (Ozone-Y_hat)
  detach(data)
  return(SSE)
}
```

- This function evaluates a linear function with an intercept and the two independent variables to compute the predicted y_hat.

- Then the algorithm computes and returns the SSE for each chromosome and we will try to minimize the SSE like OLS.

OLS Parameter Estimate with GA

- Here is the genetic algorithm for this model:

```r
ga.OLS <- ga(type='real-valued', min=c(-100,-100, -100),
             max=c(100, 100, 100), popSize=500, maxiter=500,
             names=c('intercept', 'Wind', 'Temp'),
             keepBest=T, fitness = function(b)
               -OLS(airquality, b[1],b[2], b[3]))
```

- Notice that the values are "real-based" and we specify the minimum (-100) and maximum (+100) values for each of the coefficients: β_0, β_1, β_2

- The parameters use real numbers (so floating decimals) and passes those to the linear regression equation/function.

- The SSE is 50990.17 with the GA compared to 50988.96 with the OLS approach.

```
+-----------------------------------+
|         Genetic Algorithm         |
+-----------------------------------+

GA settings:
Type                  =  real-valued
Population size       =  500
Number of generations =  500
Elitism               =  25
Crossover probability =  0.8
Mutation probability  =  0.1
Search domain
    intercept Wind Temp
Min      -100 -100 -100
Max       100  100  100

GA results:
Iterations             = 500
Fitness function value = -50990.17
Solution               =
     intercept      Wind    Temp
[1,]   -66.182 -3.310652 1.81492
```

Practical Example – The Travelling Salesman

The Traveling Salesman Problem

- One of the major topics in operations research is what is known as the “travelling salesman”. problem.

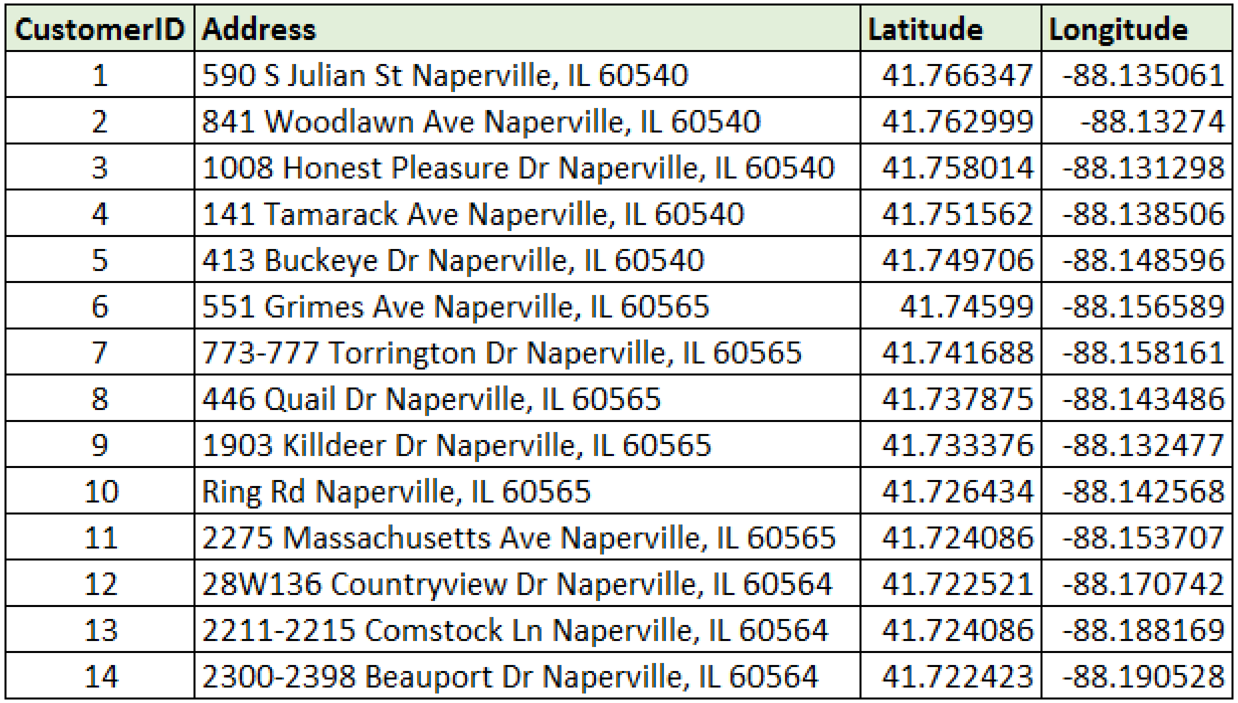
- The problem can be stated as follows:
  - There is a set of cities a salesman needs to visit and each city must be visited once.
  - What’s the shortest way through all the cities?

- This problem has real world implications (but not limited too) related to cost reductions, increase in efficiencies for deliveries, & customer satisfaction.

- Our goal is to devise:
  - A genetic algorithm to approach solving this problem.

Understanding the data

- In order to better prepare the analysis, we must first understand the data we are working with.



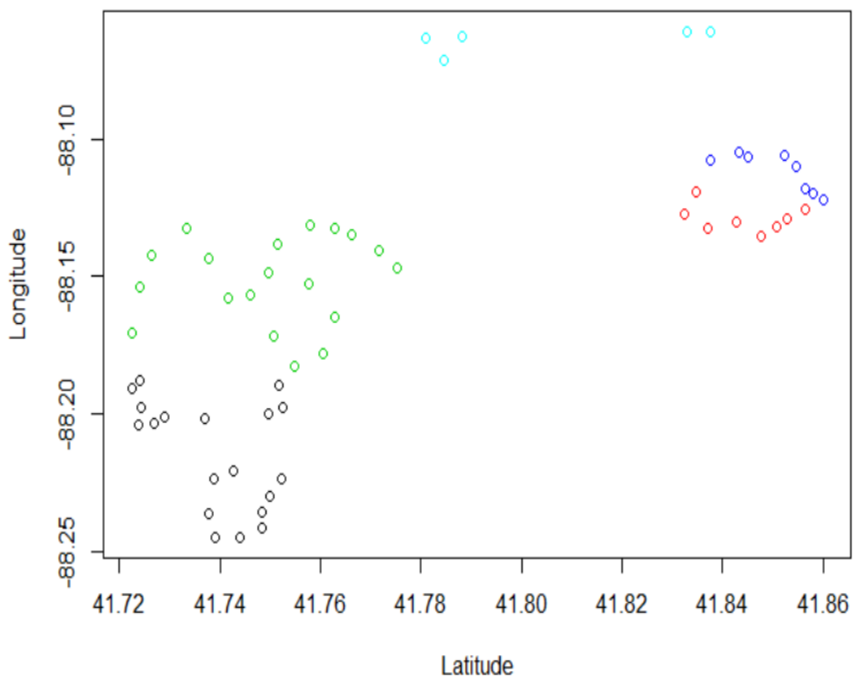
CustomerID	Address	Latitude	Longitude
1	590 S Julian St Naperville, IL 60540	41.766347	-88.135061
2	841 Woodlawn Ave Naperville, IL 60540	41.762999	-88.13274
3	1008 Honest Pleasure Dr Naperville, IL 60540	41.758014	-88.131298
4	141 Tamarack Ave Naperville, IL 60540	41.751562	-88.138506
5	413 Buckeye Dr Naperville, IL 60540	41.749706	-88.148596
6	551 Grimes Ave Naperville, IL 60565	41.74599	-88.156589
7	773-777 Torrington Dr Naperville, IL 60565	41.741688	-88.158161
8	446 Quail Dr Naperville, IL 60565	41.737875	-88.143486
9	1903 Killdeer Dr Naperville, IL 60565	41.733376	-88.132477
10	Ring Rd Naperville, IL 60565	41.726434	-88.142568
11	2275 Massachusetts Ave Naperville, IL 60565	41.724086	-88.153707
12	28W136 Countryview Dr Naperville, IL 60564	41.722521	-88.170742
13	2211-2215 Comstock Ln Naperville, IL 60564	41.724086	-88.188169
14	2300-2398 Beauport Dr Naperville, IL 60564	41.722423	-88.190528

- The classic representations of this problem do not show the data munging reshaping aspects which I feel are important to solve this problem in the real-world context. Therefore, we will spend some time with this aspect in this tutorial.

The Traveling Salesman Problem

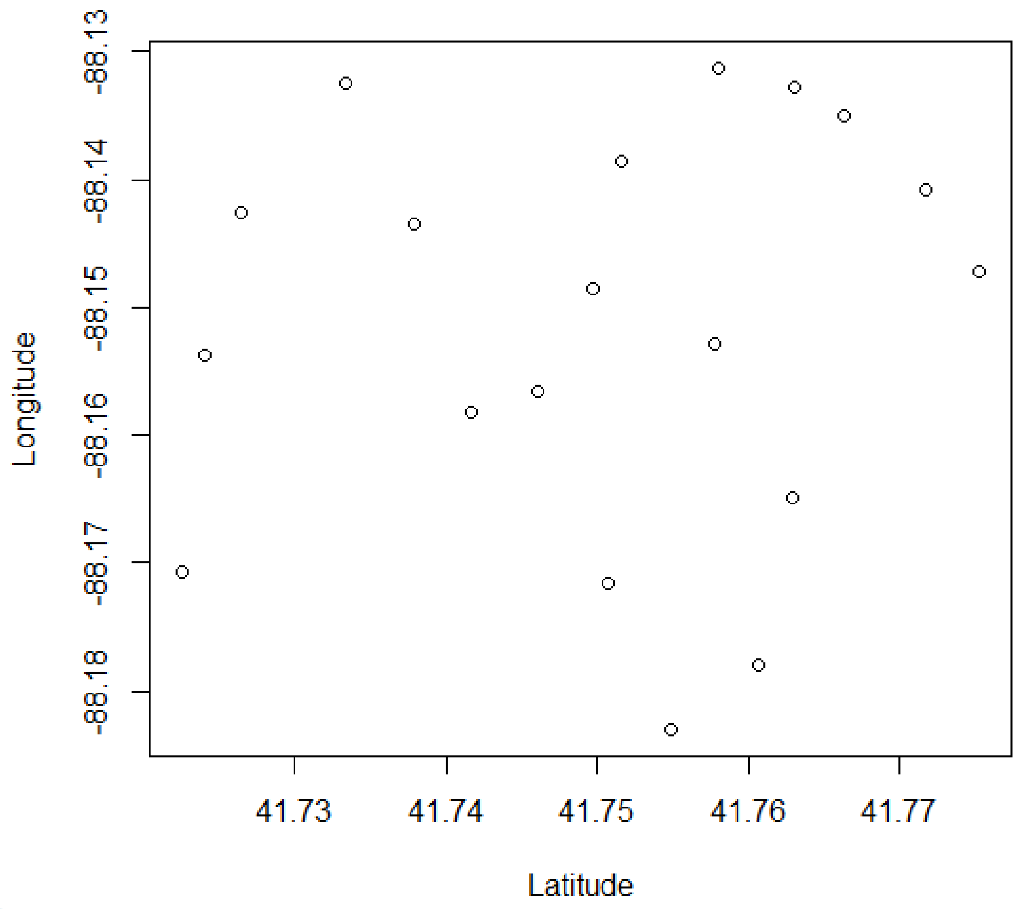
- Before we begin the example, lets say we have 5 salesmen and we want to send them to specific addresses.
- This is a prime opportunity to utilize another machine learning technique discussed earlier - clustering.
- For this example, we will use a k-means cluster approach with k = 5.
- The results of our clustering when plotted are shown on the right hand side.

K-Means Cluster



The Traveling Salesman Problem

- Lets focus our analysis on a single cluster (k=3)



The Traveling Salesman Problem

- We have to transform the latitude and longitude coordinates between each of the points into a distance matrix before we can continue.

```r
# Remove unecessary clusters

TSPdata$Address <- NULL
TSPdata$Cluster <- NULL
# TSPdata$CustomerID <- NULL

# This section require the GeoGraphic Function to be executed.

colnames(TSPdata)[1] <- "name"
colnames(TSPdata)[2] <- "lat"
colnames(TSPdata)[3] <- "lon"

# Run the GeoGraphic Distance Function.R

Distmatrix <- GeoDistanceInMetresMatrix(TSPdata)
D <- as.matrix(Distmatrix)
```

```r
ReplaceLowerOrUpperTriangle <- function(m, triangle.to.replace){
  if (nrow(m) != ncol(m)) stop("Supplied matrix must be square.")
  if      (tolower(triangle.to.replace) == "lower") tri <- lower.tri(m)
  else if (tolower(triangle.to.replace) == "upper") tri <- upper.tri(m)
  else stop("triangle.to.replace must be set to 'lower' or 'upper'.")
  m[tri] <- t(m)[tri]
  return(m)
}

GeoDistanceInMetresMatrix <- function(df.geopoints){
  GeoDistanceInMetres <- function(g1, g2){
    DistM <- function(g1, g2){
      require("Imap")
      return(ifelse(g1$index > g2$index, 0, gdist(lat.1=g1$lat,
                    lon.1=g1$lon, lat.2=g2$lat, lon.2=g2$lon, units="m")))
    }
    return(mapply(DistM, g1, g2))
  }
  n.geopoints <- nrow(df.geopoints)
  df.geopoints$index <- 1:n.geopoints
  list.geopoints <- by(df.geopoints[,c("index", "lat", "lon")], 1:n.geopoints,
                     function(x){return(list(x))})
  mat.distances <- ReplaceLowerOrUpperTriangle(outer(list.geopoints,
                       list.geopoints, GeoDistanceInMetres), "lower")

  rownames(mat.distances) <- df.geopoints$name
  colnames(mat.distances) <- df.geopoints$name

  return(mat.distances)
}
```

The Traveling Salesman Problem

- This is the resulting distance matrix, shown in meters.

	Customer 1	Customer 2	Customer 3	Customer 4	Customer 5	Customer 6	Customer 7
Customer 1	0	418.9615	977.0045	1666.9535	2164.0731	2884.0543	3345.4845
Customer 2	418.9615	0	566.5145	1357.7877	1979.5903	2739.1884	3173.7743
Customer 3	977.0045	566.5145	0	934.2948	1709.1605	2491.6294	2877.5394
Customer 4	1666.9535	1357.7877	934.2948	0	864.1673	1626.4153	1968.6405
Customer 5	2164.0731	1979.5903	1709.1605	864.1673	0	782.5283	1194.184
Customer 6	2884.0543	2739.1884	2491.6294	1626.4153	782.5283	0	495.3855
Customer 7	3345.4845	3173.7743	2877.5394	1968.6405	1194.184	495.3855	0

The Traveling Salesman Problem

- The fitness function to be maximized can be defined as the reciprocal of the tour length.

```r
tourLength <- function(tour, distMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour,2)[,2:1]
  sum(distMatrix[route])
}

tspFitness <- function(tour, ...) 1/tourLength(tour, ...)
```

- We then run the genetic algorithm in the GA package specifying the type = "permutation".

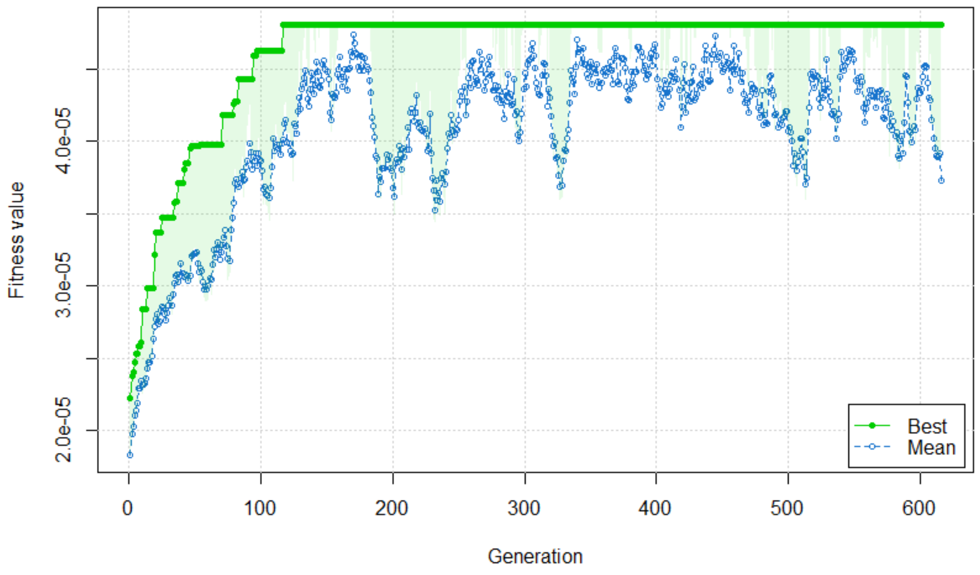
```r
GA <- ga(type= "permutation", fitness= tspFitness, distMatrix=D, min=1,
         max=nrow(D), popSize=50, maxiter = 5000, run=500, pmutation=0.2)
```

The Traveling Salesman Problem

- Here is the output of the algorithm:

```
+-----------------------------------+
|         Genetic Algorithm         |
+-----------------------------------+

GA settings:
Type                  =  permutation
Population size       =  50
Number of generations =  5000
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.2

GA results:
Iterations             = 616
Fitness function value = 4.804968e-05
```

Genetic Algorithm Evolution Chart

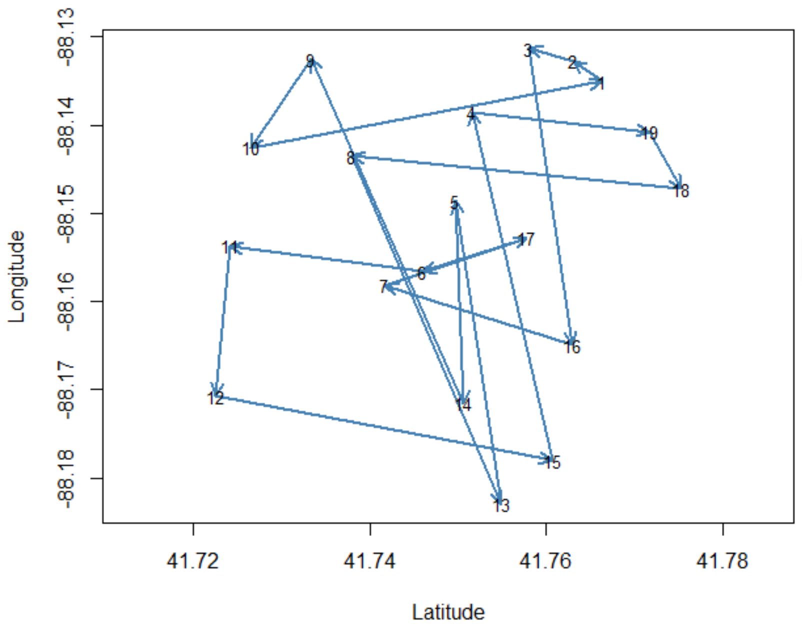


- The solution that corresponds to the unique path can be shown by:

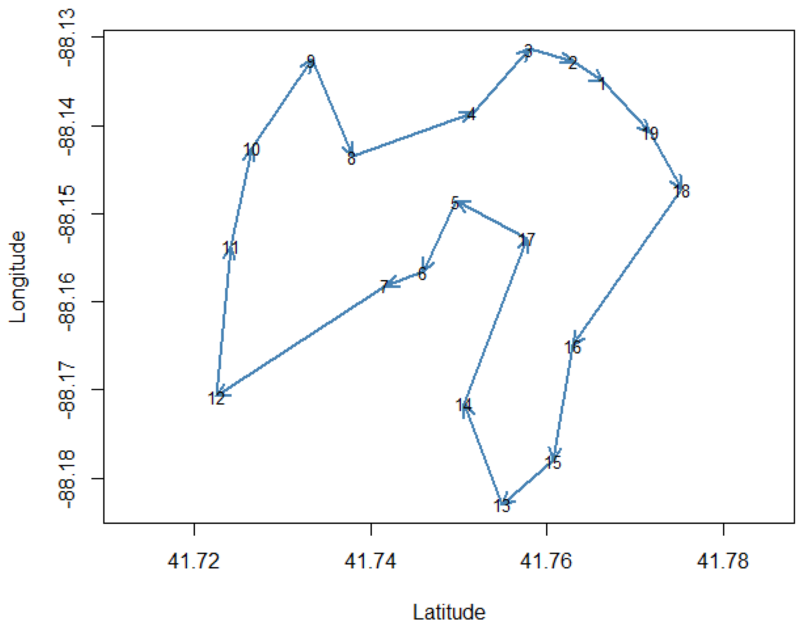
```
apply(GA@solution, 1, tourLength, D)
```

The Traveling Salesman Problem

1st Route



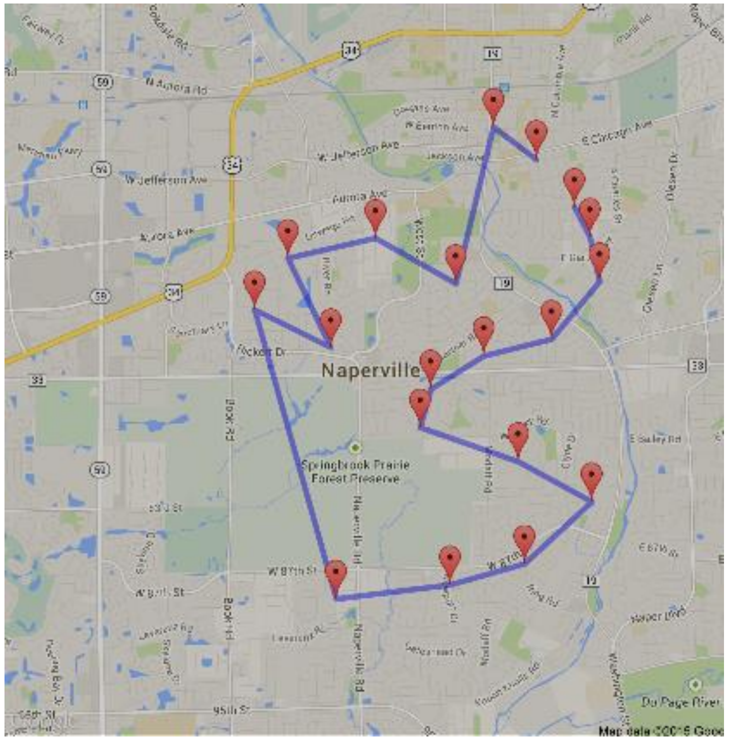
Final route optimized by the Genetic Algorithm



```
# This solution will show the map of the ideal route produced by the GA.

x <- TSPdata[, 2]
y <- TSPdata[, 3]
plot(x,y, type= "n", asp=1, xlab="Latitude", ylab="Longitude", col="green")
tour <- GA@solution[1,]
tour <- c(tour, tour[1])
n <- length(tour)
abline(arrows(x[tour[-n]], y[tour[-n]], x[tour[-1]],
              y[tour[-1]], length=0.15, angle=25, col = "steelblue", lwd=2))
text(x,y, labels(TSPdata$name), cex=0.8)
```

The Traveling Salesman Problem

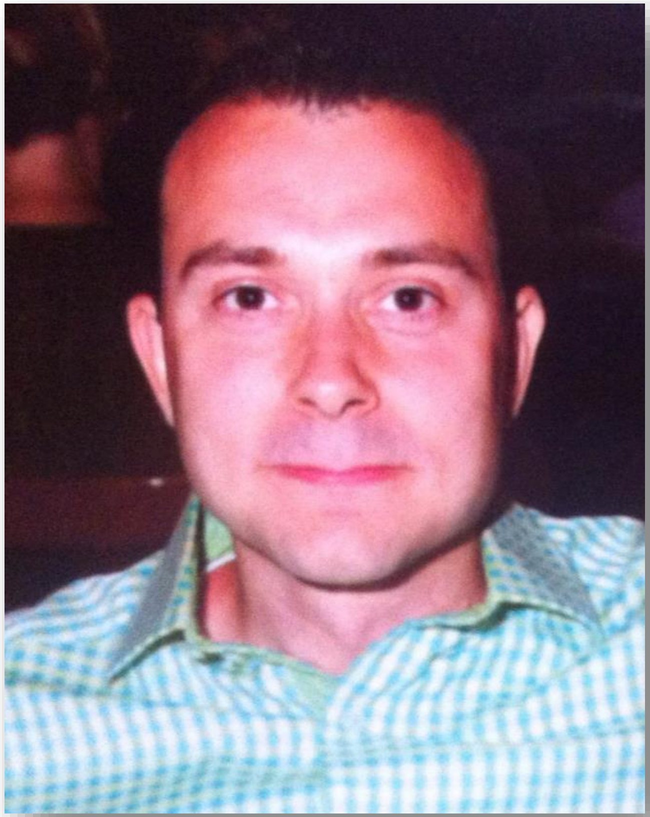
Here is the optimal solution depicted on the a google map.



The solution passed the results from R through the google api via the library(ggmap).

About Me

- Reside in Wayne, Illinois
- Active Semi-Professional Classical Musician (Bassoon).
- Married my wife on 10/10/10 and been together for 10 years.
- Pet Yorkshire Terrier / Toy Poodle named Brunzie.
- Pet Maine Coons' named Maximus Power and Nemesis Gul du Cat.
- Enjoy Cooking, Hiking, Cycling, Kayaking, and Astronomy.
- Self proclaimed Data Nerd and Technology Lover.



Acknowledgements

- http://www.genetic-programming.org/
- http://www.slideshare.net/AMedOs/introduction-to-genetic-algorithms-26956618?related=3
- http://www.slideshare.net/deg511/genetic-algorithms-in-search-optimization-and-machine-learning?related=3
- http://www.slideshare.net/kancho/genetic-algorithm-by-example?related=4
- http://www.slideshare.net/karthiksankar/genetic-algorithms-3626322?related=6
- http://www.r-bloggers.com/genetic-algorithms-a-simple-r-example/
- http://stats.seandolinar.com/genetic-algorithm-to-minimize-ols-regression-in-r/

Fine