Previamente

- Problemas P que pueden ser resueltos tiempo polinomial
- Problemas NP soluciones positivas/afirmativas pueden ser verificadas en tiempo polinomial
- Problemas NPC en tiempo exponencial
  - Problema del viajero

Problema del viajero

- El mejor algoritmo que se conoce para resolver este problema es $O(n^2 2^N)$.
- El tamaño del espacio de búsqueda del problema del viajero crece conforme a la expresión: $(n-1)! / 2$
- Algunos ejemplos de lo que esto significa, son los siguientes:
- Para n = 10, hay unas 181,000 soluciones posibles.
- Para n = 20, hay unas 10,000,000,000,000,000 soluciones posibles.
- Para n = 50, hay unas 100,000,000,000,000,000,000,000,000,000, 000,000, 000,000,000,000,000,000,000,000,000,000 soluciones posibles.
- Para tener idea de estas magnitudes, basta decir que sólo hay 1,000,000,000,000,000,000,000 litros de agua en el planeta

Optimización

Técnicas clásicas de optimización:

- Métodos como el del gradiente conjugado requieren de la primera derivada de la función objetivo.
- Otros, como el de Newton, requieren además de la segunda derivada.
- Por tanto, si la función objetivo no es diferenciable (y en algunos problemas del mundo real, ni siquiera está disponible en forma explícita)

Heurística

- Cuando enfrentamos espacios de búsqueda tan grandes como en el caso del problema del viajero, y que además los algoritmos más eficientes que existen para resolver el problema requieren tiempo exponencial, resulta obvio que las técnicas clásicas de búsqueda y optimización son insuficientes. Es entonces cuando recurrimos a las “heurísticas”.

Heurística

- La palabra “heurística” se deriva del griego *heuriskein*, que significa “encontrar” o “descubrir”.
- Newell et al. [173] dicen:
- A un proceso que puede resolver un cierto problema, pero que no ofrece ninguna garantía de lograrlo, se le denomina una ‘heurística’ para ese problema.

Heurística

- Una definición más precisa y adecuada para los fines de este curso es la proporcionada por Reeves [186]:

- Una heurística es una técnica que busca soluciones buenas (es decir, casi óptimas) a un costo computacional razonable, aunque sin garantizar factibilidad u optimalidad de las mismas. En algunos casos, ni siquiera puede determinar qué tan cerca del óptimo se encuentra una solución factible en particular..

Heurística

- Algunos ejemplos de técnicas heurísticas son los siguientes:
- Búsqueda Tabú
- Recocido Simulado
- Colonia de abejas
- Enjambre de Partículas
- Búsqueda armónica
- Algoritmos genéticos

Genetic Algorithms

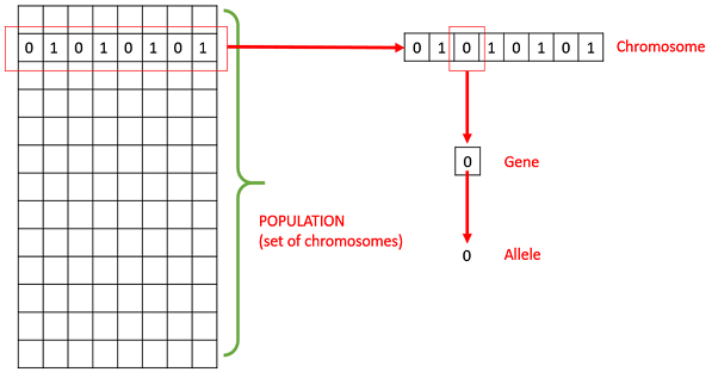
GA Quick Overview

- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete optimization
- Attributed features:
  - not too fast
  - good heuristic for combinatorial problems
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

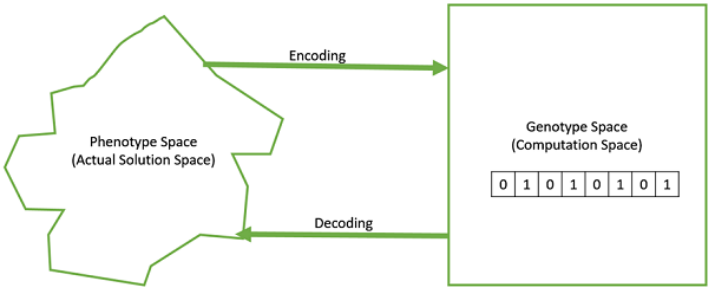
Genetic algorithms

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

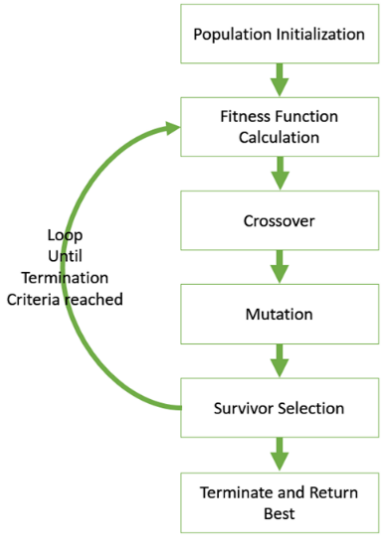
Basic Terminology



Basic Terminology



Basic Structure



Basic Terminology

- **Fitness Function** − A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.

- **Genetic Operators** − These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

SGA technical summary tableau

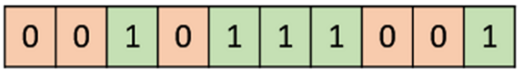
Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

Generalized pseudo- code for a GA

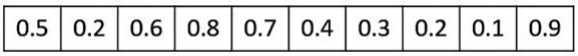
```
GA()
   initialize population
   find fitness of population

   while (termination criteria is reached) do
      parent selection
      crossover with probability pc
      mutation with probability pm
      decode and fitness calculation
      survivor selection
      find best
   return best
```

Representation

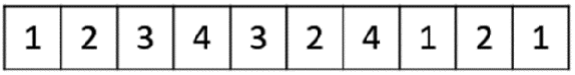
- Binary



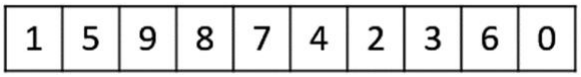
- Real Values



- Integer



- Permutation



Initialization

- **Random Initialization** − Populate the initial population with completely random solutions.
- **Heuristic initialization** − Populate the initial population using a known heuristic for the problem.

Population Models

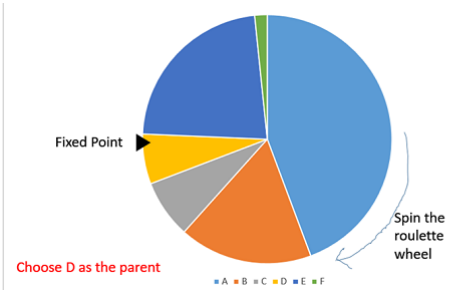
- Steady State
  - In steady state GA, we generate one or two off-springs in each iteration and they replace one or two individuals from the population. A steady state GA is also known as **Incremental GA**.
- Generational
  - In a generational model, we generate ‘n’ off-springs, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.

Fitness function

- The fitness function simply defined is a function which takes a **candidate solution to the problem as input and produces as output** how “fit” our how “good” the solution is with respect to the problem in consideration.
- A fitness function should possess the following characteristics:
  - The fitness function should be sufficiently fast to compute.
  - It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

Roulette Wheel Selection

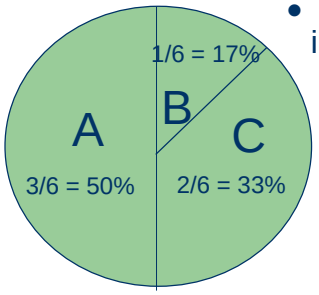
- Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.
- Two implementations of fitness proportionate selection are possible



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Roulette Wheel Selection

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



fitness(A) = 3

fitness(B) = 1

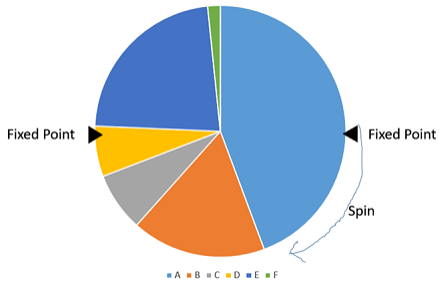
fitness(C) = 2

Fitness proportionate selection

- Implementation wise, we use the following steps −
- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

Stochastic Universal Sampling (SUS)

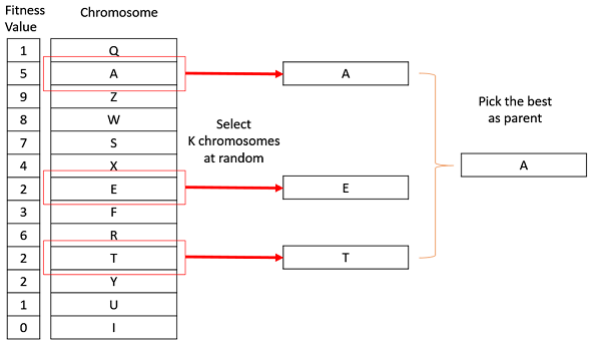
- Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once. It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

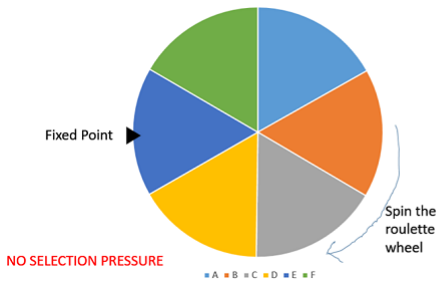
Tournament Selection

- In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Rank Selection

- Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run). This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

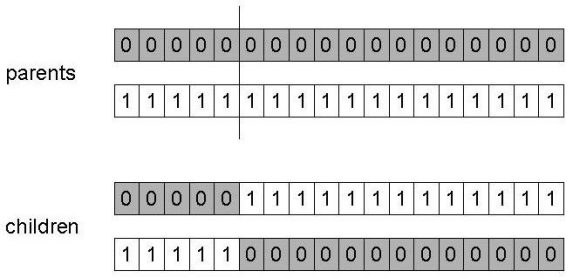
Rank Selection

- In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

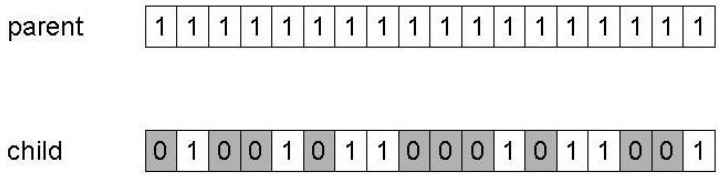
SGA operators: 1-point crossover

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



SGA operators: mutation

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



The simple GA

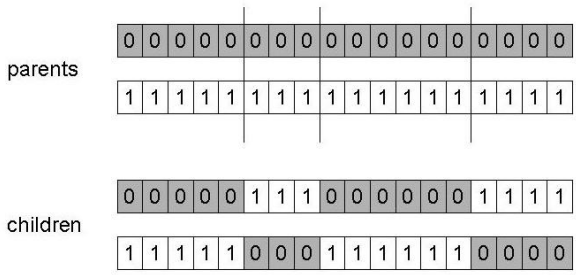
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations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

Alternative Crossover Operators

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

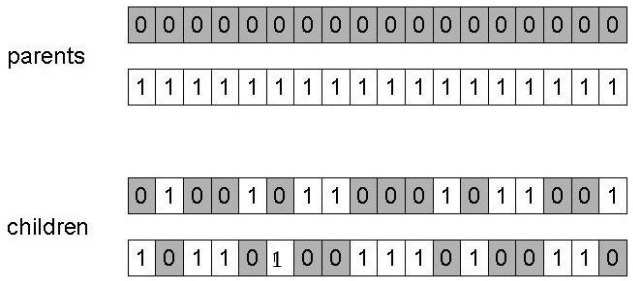
n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents



Uniform crossover

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



Crossover OR mutation?

- Decade long debate: which one is better / necessary / main-background

- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role

Crossover OR mutation? (cont'd)

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

Crossover OR mutation? (cont'd)

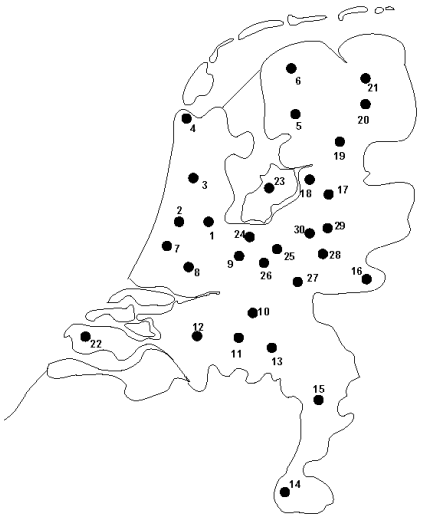
- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of To hit the optimum you often need a 'lucky' mutation

Permutation Representations

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

Permutation representation: TSP example

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

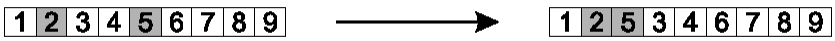


Mutation operators for permutations

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

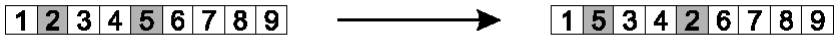
Insert Mutation for permutations

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



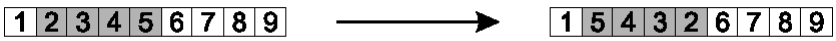
Swap mutation for permutations

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



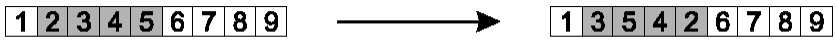
Inversion mutation for permutations

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



Scramble mutation for permutations

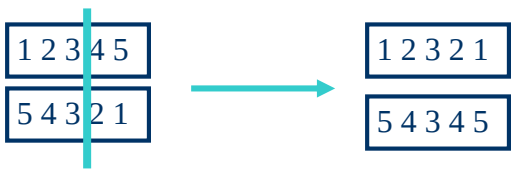
- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



(note subset does not have to be contiguous)

Crossover operators for permutations

- "Normal" crossover operators will often lead to inadmissible solutions



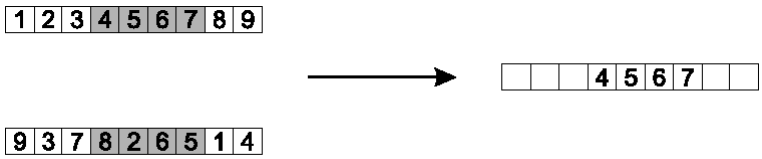
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Order 1 crossover

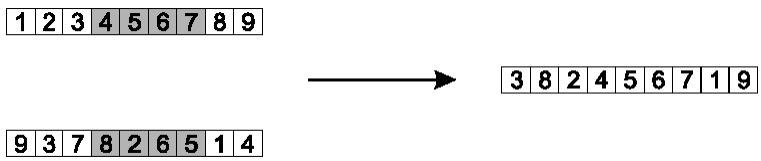
- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,
     - using the **order** of the second parent
     - and wrapping around at the end
  4. Analogous for the second child, with parent roles reversed

Order 1 crossover example

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



Partially Mapped Crossover (PMX)

Informal procedure for parents P1 and P2:

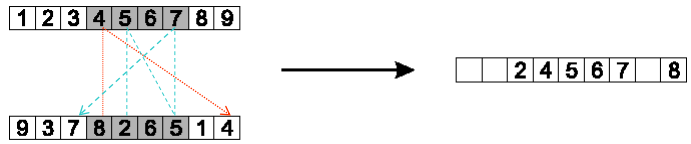
1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied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

Second child is created analogously

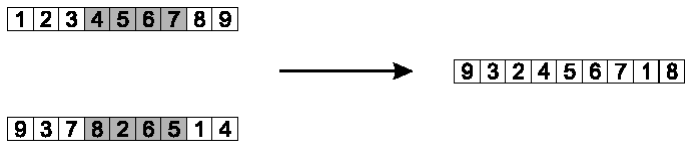
PMX example

- Step 1

- Step 2



- Step 3



Cycle crossover

Basic idea:

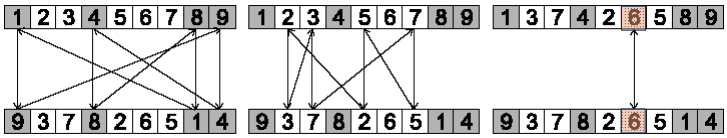
Each allele comes from one parent *together with its position*.

Informal procedure:

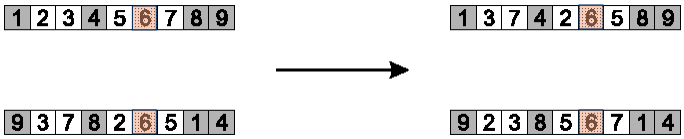
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

Cycle crossover example

- Step 1: identify cycles



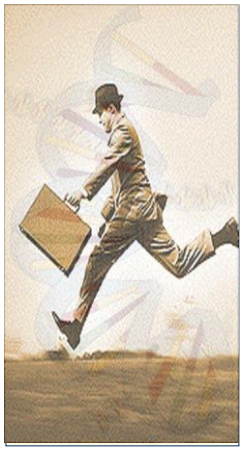
- Step 2: copy alternate cycles into offspring



Knapsack Problem

Problem Description:

- You are going on a picnic.
- And have a number of items that you could take along.
- Each item has a weight and a benefit or value.
- You can take one of each item at most.
- There is a capacity limit on the weight you can carry.
- You should carry items with max. values.



Knapsack Problem

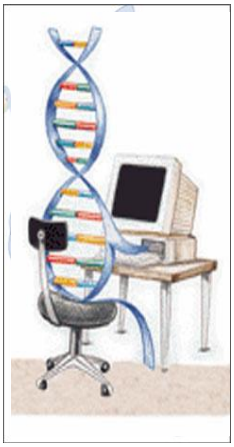
Example:

- **Item: 1 2 3 4 5 6 7**
- **Benefit: 5 8 3 2 7 9 4**
- **Weight: 7 8 4 10 4 6 4**
- **Knapsack holds a maximum of 22 pounds**
- **Fill it to get the maximum benefit**

Genetic Algorithm

Outline of the Basic Genetic Algoritm

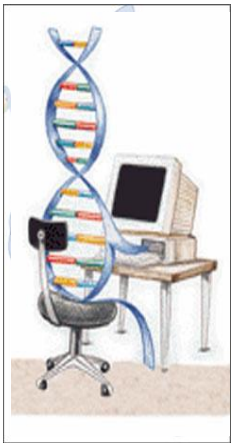
1. **[Start]**
   - ✓ Encoding: represent the individual.
   - ✓ Generate random population of *n* chromosomes (suitable solutions for the problem).
2. **[Fitness]** Evaluate the fitness of each chromosome.
3. **[New population]** repeating following steps until the new population is complete.
4. **[Selection]** Select the best two parents.
5. **[Crossover]** cross over the parents to form a new offspring (children).



Genetic Algorithm

Outline of the Basic Genetic Algoritm Cont.

6. **[Mutation]** With a mutation probability.
7. **[Accepting]** Place new offspring in a new population.
8. **[Replace]** Use new generated population for a further run of algorithm.
9. **[Test]** If the end condition is satisfied, then **stop**.
10. **[Loop]** Go to step **2** .



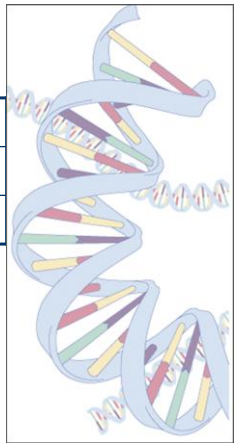
Basic Steps

Start

- Encoding: 0 = not exist, 1 = exist in the Knapsack
  Chromosome: 1010110

Item.	1	2	3	4	5	6	7
Chro	1	0	1	0	1	1	0
Exist?	y	n	y	n	y	y	n

  => Items taken: 1, 3 , 5, 6.
- Generate random population of *n* chromosomes:
  a) 0101010
  b) 1100100
  c) 0100011



Basic Steps Cont.

Fitness & Selection

a) 0101010: Benefit= 19, Weight= 24 ✘

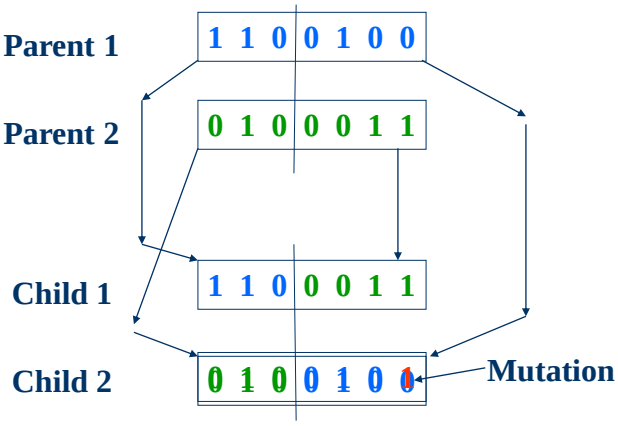
Item	1	2	3	4	5	6	7
Chro	0	1	0	1	0	1	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

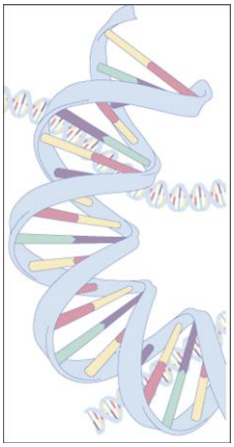
b) 1100100: Benefit= 20, Weight= 19. ✓
c) 0100011: Benefit= 21, Weight= 18. ✓

=> We select Chromosomes b & c.

Basic Steps Cont.

Crossover & Mutation

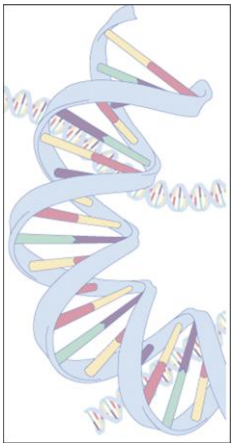




Basic Steps Cont.

Accepting, Replacing & Testing

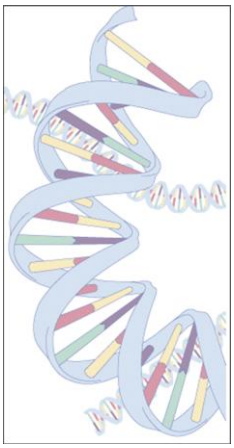
- ✓ Place new offspring in a new population.
- ✓ Use new generated population for a further run of algorithm.
- ✓ If the end condition is satisfied, then **stop**. End conditions:
  - Number of populations.
  - Improvement of the best solution.
- ✓ Else, return to step 2 **[Fitness].**



Genetic Algorithm

Conclusion

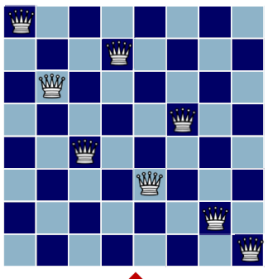
- GA is nondeterministic – two runs may end with different results
- There's no indication whether best individual is optimal



Práctica

Ejemplo: El problema de las N reinas

- Fenotipo:
  Configuración del tablero.
- Genotipo:
  Permutación de enteros.



1	3	5	2	6	4	7	8

Ejemplo: El problema de las N reinas

- Función de evaluación:
  Número de parejas de reinas que no se atacan.

  Definición alternativa como problema de minimización:
  - Penalización de una reina =
    Número de reinas a las que ataca directamente.
  - Penalización del tablero =
    Suma de las penalizaciones de todas las reinas.
  - Fitness = - penalización del tablero.

Ejemplo: El problema de las N reinas

- Operador de mutación:
  Pequeña variación en una permutación.

  Ejemplo:
  Intercambio de dos posiciones elegidas al azar.

Ejemplo: El problema de las N reinas

- Operador de cruce:
  Combinación de dos permutaciones
  - Elegir un punto de cruce al azar.
  - Copiar la primera parte de ambas.
  - Rellenar la segunda parte usando el otro padre: se añaden valores desde el punto de cruce, en el orden en el que aparecen y saltando los que ya están.