VC6 – Algoritmos genéticos

03MAIR – Algoritmos de optimización

viu Universidad Internacional de Valencia

Agenda

1. Algoritmos genéticos y evolutivos
   - Introducción conceptos evolucionistas
   - Fundamentos de los algoritmos evolutivos y genéticos
   - Características de los algoritmos evolutivos y genéticos
   - Componentes de los algoritmos evolutivos y genéticos

2. Problemas del seminario

3. Aplicación práctica del Colonia de Hormigas(ACO) al problema del agente viajero(TSP)

VC6 – Algoritmos genéticos

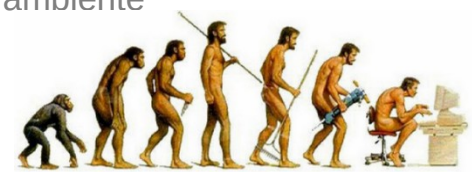
03MAIR – Algoritmos de optimización

viu Universidad Internacional de Valencia

Introducción a conceptos evolucionistas

- El mejor mecanismo conocido para resolver problemas es… el cerebro humano.
- Creacionismo – Evolucionismo
- Antes que Darwin sobre los orígenes de la especie humana:

- Georges-Louis Leclerc, Conde de Buffon (1707-1788)(*) (1 siglo antes que Darwin)
  - · primero en plantear problemas evolutivos
- James Burnett, Lord Monboddo (1714-1799) (**)
  - · primeras ideas evolucionistas
- Jean Baptiste Lamarck (1744-1829) - Lamarckismo (***)
  - · los organismos pueden adquirir nuevas características por influencia del ambiente
  - · herencia de caracteres adquiridos



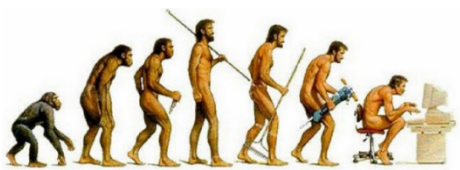
(*) https://en.wikipedia.org/wiki/Georges-Louis_Leclerc,_Comte_de_Buffon
(**) http://en.wikipedia.org/wiki/James_Burnett,_Lord_Monboddo
(***) http://en.wikipedia.org/wiki/Jean-Baptiste_Lamarck

Introducción a conceptos evolucionistas

Charles Robert Darwin, FRS (1809-1882)(*). Darwinismo

> Los individuos son ligeramente distintos entre sí y estas pequeñas variaciones hacen que cada uno tenga distintas capacidades para adaptarse a su medio ambiente, así como para reproducirse y para transmitir sus rasgos a sus descendientes.

- Con el paso del tiempo (generaciones), los rasgos de los individuos que mejor se adaptaron a las condiciones del medio ambiente se vuelven más comunes, haciendo que la población, en su conjunto, evolucione (“descendencia con modificación”).
- Del mismo modo, la naturaleza selecciona las especies mejor adaptadas para sobrevivir y reproducirse (“selección natural”).

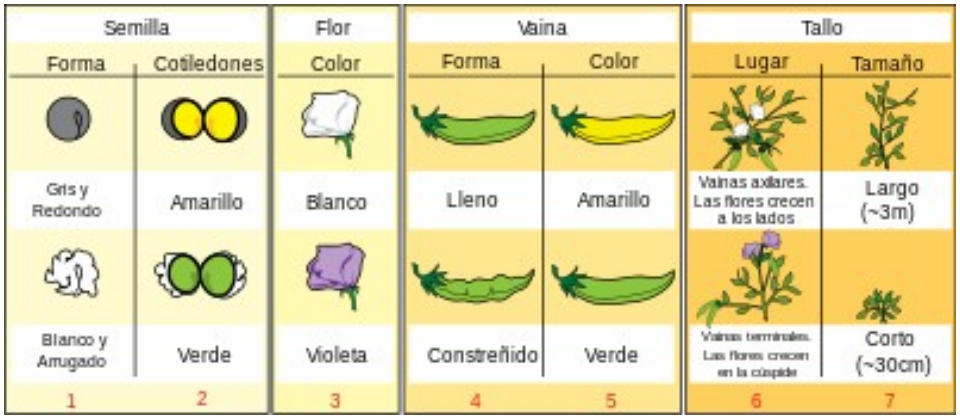


(*) http://en.wikipedia.org/wiki/Charles_Darwin

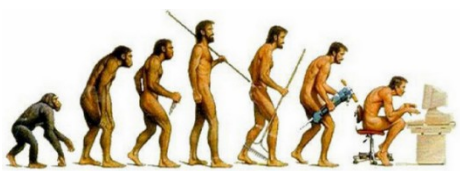
Introducción a conceptos evolucionistas

Gregor Johann Mendel (1822-1884)(*). Leyes de Mendel(**)

- Reglas básicas que gobiernan la transmisión por herencia de las características de los padres a los hijos.



Fuente: https://es.wikipedia.org/wiki/Leyes_de_Mendel

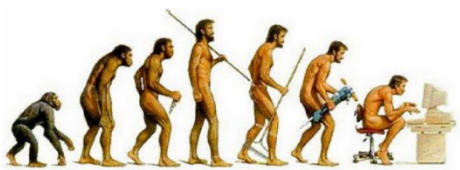


(*) http://es.wikipedia.org/wiki/Gregor_Mendel
(**) http://es.wikipedia.org/wiki/Leyes_de_Mendel

Introducción a conceptos evolucionistas

James Mark Baldwin (1861-1934)(*). El efecto Baldwin

- Las condiciones genéticas transmisibles por herencia pueden hacer **más fácil el aprendizaje** de técnicas y trucos que sólo poseen aquellos que tengan una variante evolutiva determinada.
- El comportamiento sostenido de una especie o grupo puede guiar la evolución de esa especie
- Las **habilidades** que inicialmente requieren el aprendizaje son finalmente **reemplazadas** por la evolución de sistemas genéticamente determinados **que no requieren aprendizaje**.



(*) http://en.wikipedia.org/wiki/James_Mark_Baldwin

Computación Evolutiva

Hans-Joachin Bremermann (1926-1996) (*)

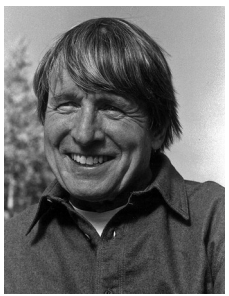
- Fue el primero en ver el componente de optimización en los procesos evolutivos .
- Primeras simulaciones con cadenas binarias(0,1) con operadores de reproducción selección, mutación.
- "**Optimization through evolution and recombination**". In: Self-Organizing systems 1962, edited M.C. Yovitts et al., Spartan Books, Washington, D.C. pp. 93–106.

(*) *https://en.wikipedia.org/wiki/Hans-Joachim_Bremermann*

Algoritmos Genéticos. GA

John Henry Holland (1929-2015)(*). Universidad de Michigan

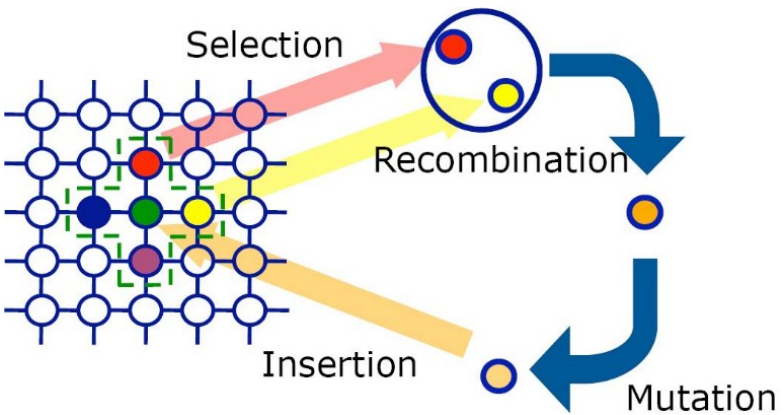
- "**Adaptation in Natural and Artificial Systems**". 1975
- Presentó el algoritmo genético como una abstracción de la evolución natural y proporcionó el marco teórico su adaptación a la computación.
- Introdujo los conceptos de población, cromosomas(cadenas binarias), evolución a través de operadores de cruce + mutación y la selección.
- TEDxUofM - John Holland - Building Blocks and Innovation: https://youtu.be/nzHVGd22vak



(*) *https://en.wikipedia.org/wiki/John_Henry_Holland*

Fundamentos de los algoritmos genéticos y evolutivos

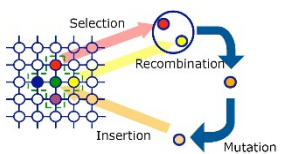
- Se hace **evolucionar** una población de individuos (cada uno de los cuales puede representar una posible solución)
- La población se somete a **acciones aleatorias** semejantes a las de la evolución biológica (mutaciones y recombinaciones genéticas).
- Los individuos se seleccionan de acuerdo con una función de adaptación(**fitness**) en función de la cual se decide qué individuos sobreviven (los más adaptados) y cuáles son descartados (los menos aptos).



Fundamentos de los algoritmos genéticos y evolutivos

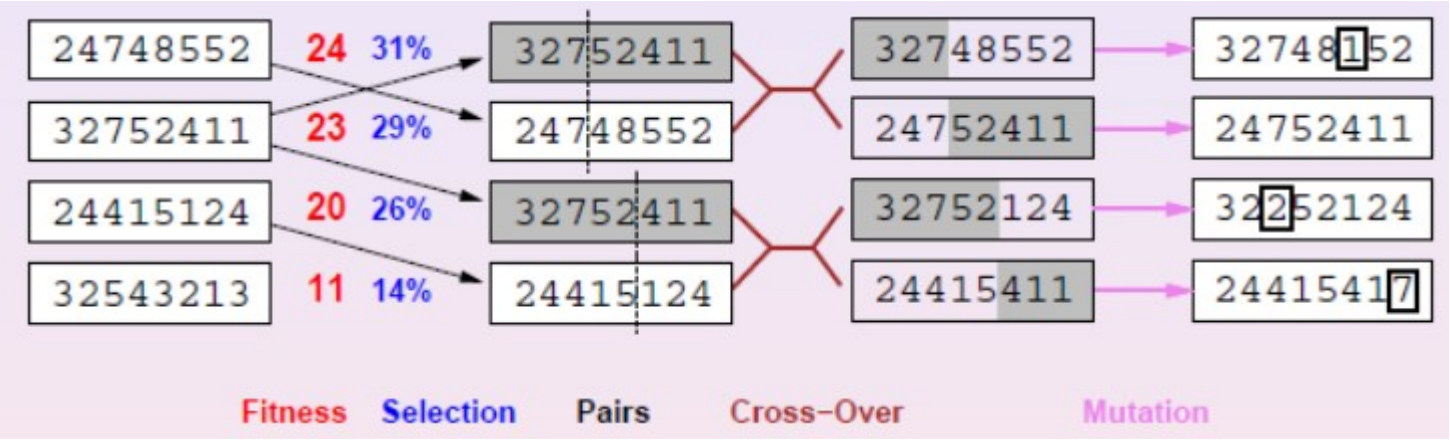
Esquema General

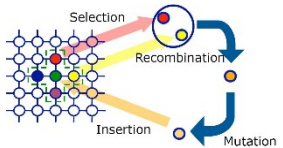
```
t ← 0
población(t) ← poblaciónInicial
EVALUAR(población(t))

while not (condición de terminación)
    t ← t + 1
    población(t) ← SELECCIONAR(población(t-1))
    población(t) ← CRUZAR(población(t))
    población(t) ← MUTAR(población(t))
    EVALUAR(población(t))

return población(t)
```



Fundamentos de los algoritmos genéticos y evolutivos

Esquema General. Selección, Cruce y Mutación





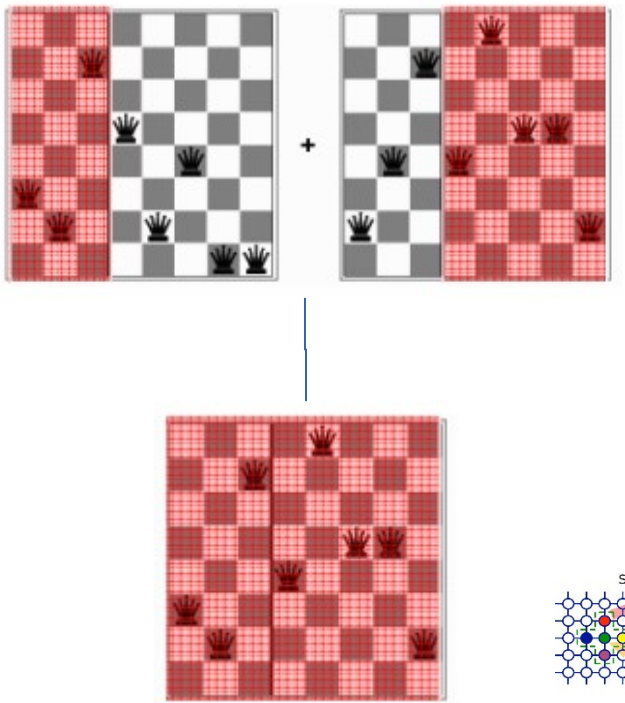
Fundamentos de los algoritmos genéticos y evolutivos

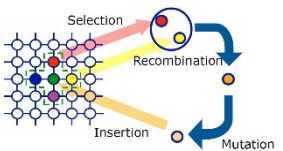
Esquema General. Selección, Cruce y Mutación

- Ejemplo 8 reinas:

  - Función de evaluación
    Cantidad de parejas que no se amenazan

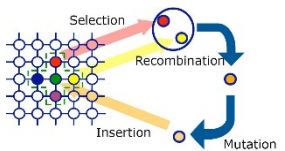
  - Operador de cruce





Características de los algoritmos genéticos y evolutivos

- Los algoritmos genéticos y evolutivos se utilizan para solucionar **problemas complejos**, pero **no garantizan la optimalidad**(técnicas heurísticas)
- En muchas ocasiones, ni siquiera podemos determinar cómo de cerca.
- Usan poblaciones de soluciones potenciales(en vez de un solo individuo), lo que hace menos probable de quedar **atrapadas en óptimos locales**.
- No necesitan disponer de **conocimiento específico** sobre el problema que intentan resolver (aunque puede ayudar).
- Usan **operadores probabilistas**, mientras las técnicas tradicionales(algoritmos exactos) utilizan operadores determinísticos.
- Se adaptan bien a problemas dinámicos(circunstancias cambiantes en el tiempo)



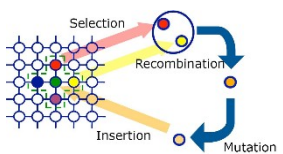
Características de los algoritmos genéticos y evolutivos

- Aunque las técnicas evolutivas sean estocásticas, el hecho de que usen operadores probabilísticos no significa que operen de manera análoga a una simple búsqueda aleatoria: la función de **fitness** ayuda a guiar el proceso de búsqueda.

- Suelen funcionar mejor(rendimiento aceptable) que los algoritmos genéricos de búsqueda pero puede haber casos en los que no se comporten bien(eficiencia)

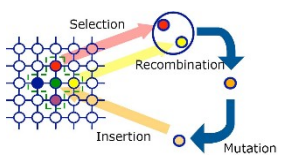
- Podemos usarlos en:
  - Problemas que no pueden resolverse de forma exacta en tiempo polinómico (**clase NP**).

  - Problemas en los que ni siquiera podemos establecer si existe una **solución eficiente**.



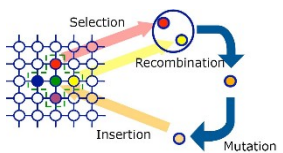
Características de los algoritmos genéticos y evolutivos

- Simplicidad. “**Fáciles” de implementar**
- Generalidad. Aplicables a **muchos ámbitos**
- Posibilidad de **incorporar conocimiento** sobre el problema
- Posibilidad de “integrarse” con **otras técnicas**
- **Paralelizables**



Componentes de un algoritmo genético/evolutivo

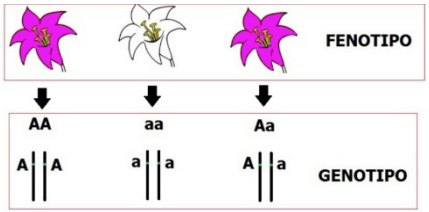
- **Representación** de las soluciones del problema.
- Función de evaluación (**fitness**/aptitud).
- Mecanismo de creación de la **población inicial**.
- **Operadores genéticos** (cruce & mutación).
- Mecanismo de **selección** [probabilística].



Componentes. Representación de soluciones

Genética Básica

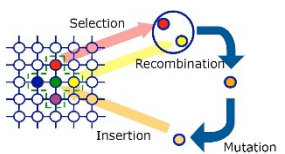
- La información necesaria para la “construcción” de un ser vivo viene codificada en su **genoma** (ADN).



- **El genotipo determina el fenotipo**.

- La correspondencia entre genes y rasgos fenotípicos es compleja:
  - un gen puede afectar a muchos rasgos.
  - muchos genes pueden afectar a un rasgo.

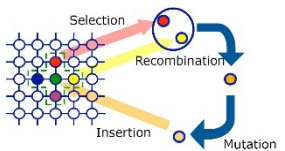
- Pequeños cambios en el genotipo pueden provocar pequeños cambios en el fenotipo del organismo (p.ej. altura, color de pelo, color de ojos…).



Componentes. Representación de soluciones

Elección sobre la representación adecuada

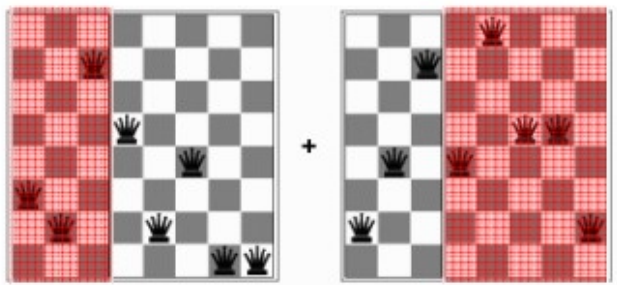
- **Diversas posibilidades** de representación.
- Elegir una que se **adapte** bien al problema:
  - los operadores de cruce y mutación deben ser consistentes con la representación
  - la selección debe basarse en el valor fitness(función de evaluación)
- Todas las soluciones deben tener una **representación**
- Las soluciones candidatas(individuos)
  - se **evalúan** de acuerdo a su **fenotipo**
  - se **codifican** según el **genotipo**



Componentes. Representación de soluciones. Ejemplo

Fenotipo

Padre Madre

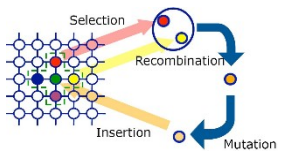


Representación

Genotipo

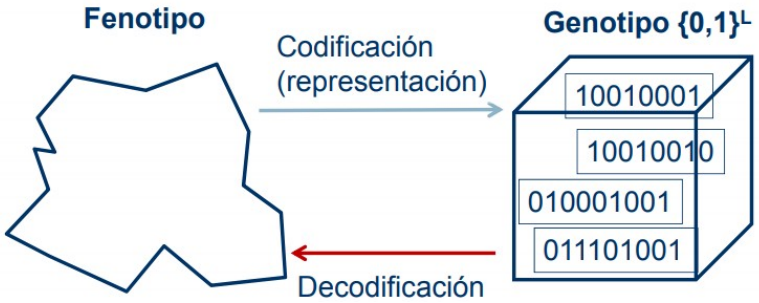
Padre : [6,7,2,4,7,5,8,8]

Madre : [7,5,2,5,1,4,4,7]



Componentes. Representación de soluciones

Representación binaria.



Representación

Cromosoma:

Cadena de bits que representa a cada individuo de la población.

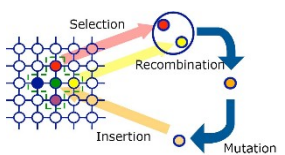
Chromosome 1	1 1 0 1 0 0 0 1 1 0 1 0
Chromosome 2	0 1 1 1 1 1 1 1 1 1 0 0

Gen:

Cada posición de la cadena.

Alelo:

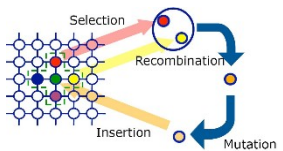
Valor de un gen.



Componentes. Función de evaluación(fitness)

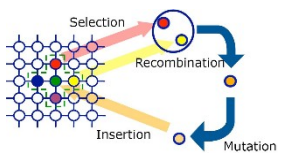
- Es la función objetivo de nuestro problema
- Asigna valores reales(*) a cada fenotipo
- Es la base del proceso de selección
- Cuanto mas eficiente sea para discriminar entre soluciones, mejor.
- Minimizar / Maximizar. Problemas equivalentes.

(*) se refiere a valores numéricos en los números reales $\mathbb{R}$



Componentes. Población inicial

- La población inicial suele ser aleatoria.
- Conviene asegurar que sea una población variada: que estén representados todos los posibles alelos para todos los genes.
- Pueden incorporarse a la población inicial individuos(soluciones) obtenidas por otros métodos(otras heurísticas) o que sepamos que tienen buenos componentes genéticos.



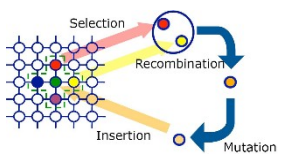
Componentes. Operadores genéticos

- Permiten la generación de nuevas soluciones(individuos) a partir de la población.

- Tipos:

  - **mutación**: modificación al azar sobre un individuo de la población.

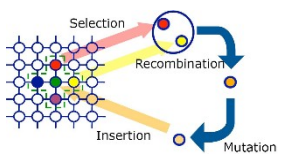
  - **cruce**: combinación de dos (o más) individuos de la población para dar lugar a otros individuos.



Componentes. Operadores genéticos

Mutación

- Los operadores de mutación actúan **sobre el genotipo** de un individuo(solución).
- Es necesario que sean aleatorios.
- Introducen diversidad genética.
- En **algoritmos genéticos** se utilizan en combinación con operadores de cruce para aumentar la diversidad.
- En **algoritmos evolutivos** son los únicos operadores que producen variedad en la población.

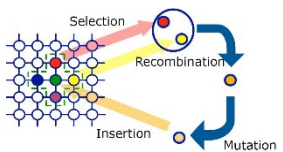


Componentes. Operadores genéticos

Cruce

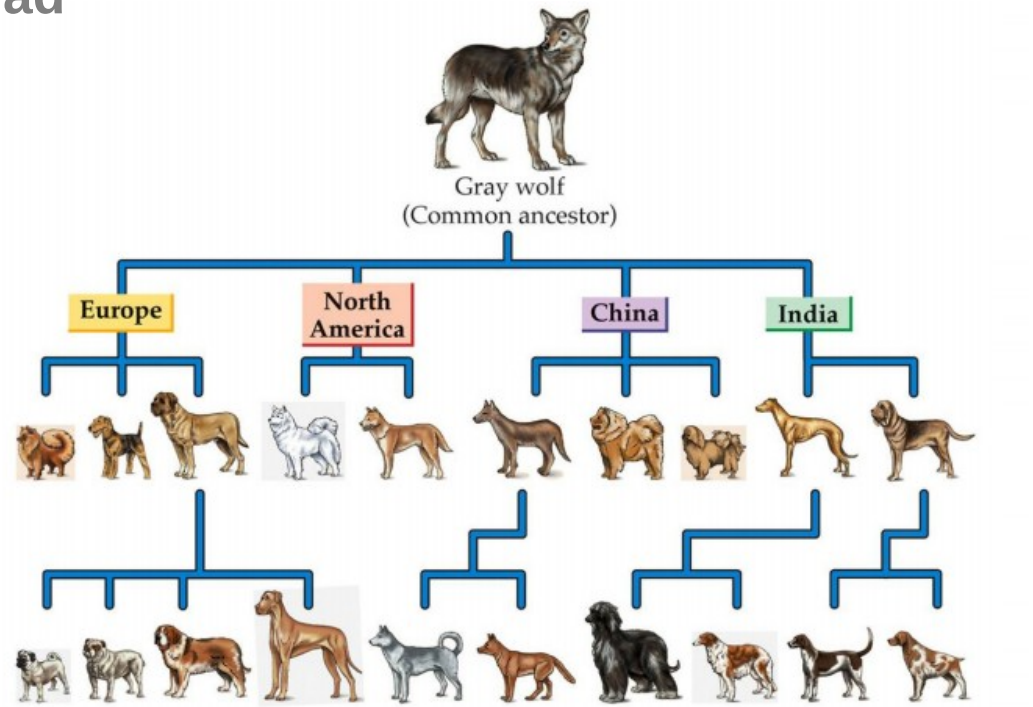
- Los operadores de cruce seleccionan los genotipos de los padres para **crear nuevos individuos**(soluciones).
- La información de los padres que se combina se elige de forma aleatoria.
- Los descendientes pueden ser peores que los padres según la función de fitness(evaluación).
- Se espera que algunos descendiente mejoren la población(*).

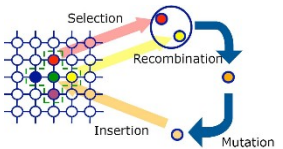
(*) El concepto se ha utilizado históricamente en la agricultura y ganadería sin tener conocimientos de genética.



Componentes. Operadores genéticos

Cruce. Variedad





Componentes. Operadores genéticos. Cruce

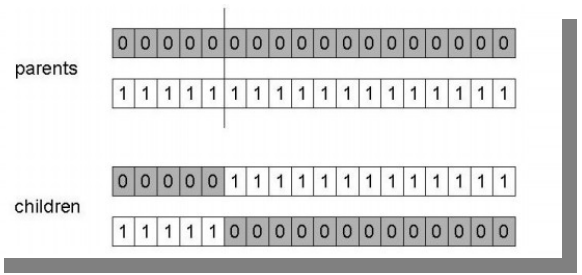
Cruce en un punto (1-point crossover)

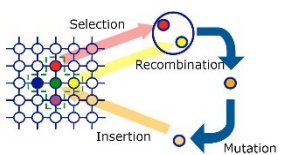
Proceso:

- se selecciona aleatoriamente un punto en la secuencia del cromosoma .
- se dividen los padres en ese punto.
- se crean hijos intercambiando partes de los cromosomas de los padres.

Limitaciones: depende del orden en el que aparecen los genes

- es más probable que sigan juntos genes próximos
- no se juntan los genes de los extremos



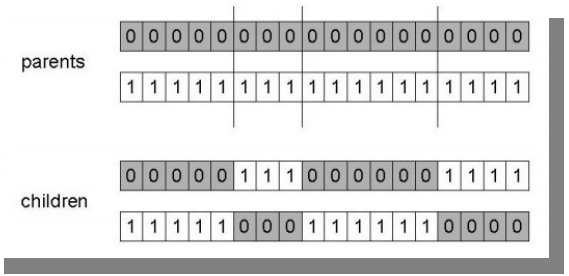


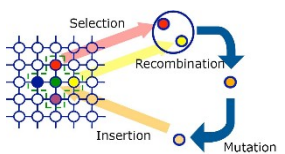
Componentes. Operadores genéticos. Cruce

Cruce en n puntos (n-point crossover)

Proceso:

- se selecciona aleatoriamente n puntos en la secuencia del cromosoma .
- se dividen los cromosomas de los padres en esos puntos.
- se crean hijos intercambiando partes de los cromosomas de los padres.





Componentes. Operadores genéticos. Cruce

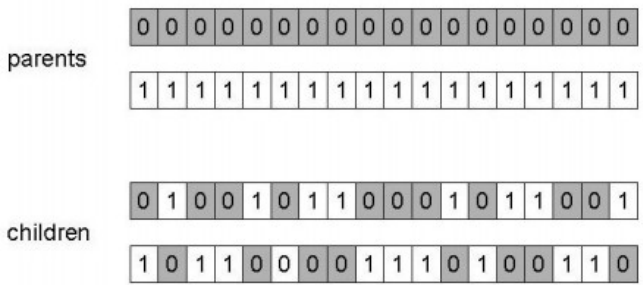
Cruce en uniforme (uniform crossover)

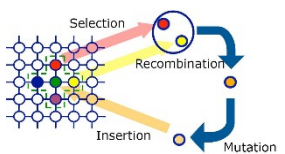
Proceso:

- se selecciona aleatoriamente el padre para cada gen.
- se crea un hijo con la selección obtenida
- se crea un nuevo hijo con la selección inversa

Ventajas:

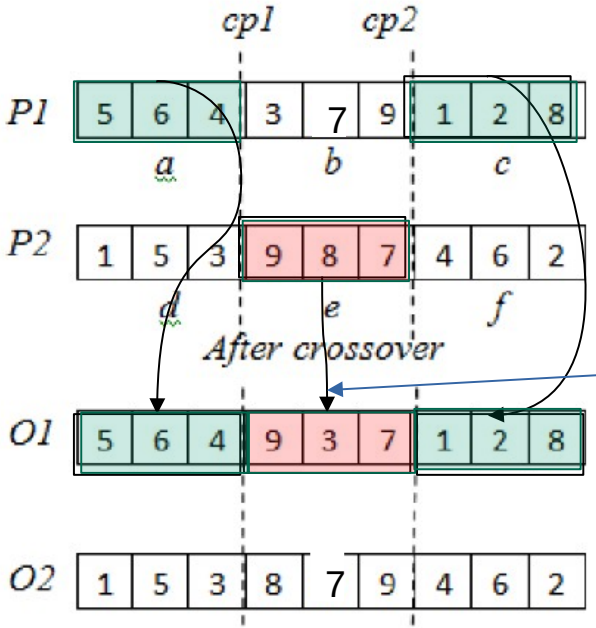
- se elimina es sesgo posicional del cruce de n punto





Componentes. Operadores genéticos. Cruce

Ejemplo de cruce para el problema del agente viajero (TSP)



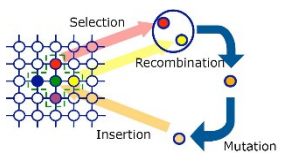
Asegurar la factibilidad

Componentes. Operadores genéticos. Cruce

Recombinación

- Operadores de cruce basados en más de dos padres(*).
- No se dan en la naturaleza pero podemos usarlos si resultan eficientes.
- Tipos:
  - basados en las frecuencias de los alelos (generalización del cruce uniforme)
  - basados en cruce de n puntos(cruce diagonal)
  - basados en operaciones numéricas sobre los alelos

(*) ¿Qué significa un bebé concebido con “tres padres”?
https://www.cra.barcelona/bebe-con-tres-padres/



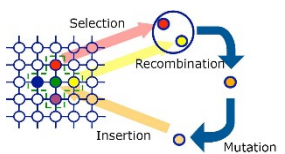
Componentes. Operadores genéticos.

Cruce vs Mutación

- En general suele ser buena estrategia **usar ambos**.
- Si prescindimos del cruce, son **algoritmos evolutivos**.

- El operador **cruce**
  - permite explorar espacios de búsqueda muy diferentes de los padres(**Diversificación**)
  - permite combinar lo mejor de los padres

- El operador **mutación** explora el espacio cercano a una solución(**Intensificación**)



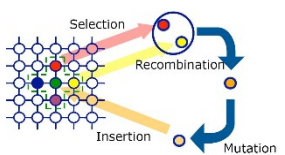
Componentes. Mecanismos de selección.

- Analogía biológica:
  - los individuos tienen instintos de **reproducción**
  - los individuos compiten por **recursos limitados**
  - es inevitable una selección basada en los **mejor adaptados**

- Reproducción: **operadores genéticos**

- Recursos limitados: mantener la **población estable** en número

- Mejor adaptados: **Mecanismo de selección**



Componentes. Mecanismos de selección.

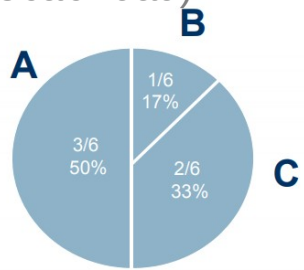
- Procedimiento:

  - se **asignan probabilidades** de selección a los individuos según su función de fitness(evaluación)

  - se aplica el **proceso aleatorio(*)**:

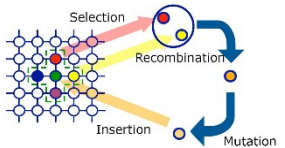
    . los mejores individuos son más probables de ser elegidos [**ruleta**(**)]

    . los peores individuos también pueden ser elegidos(pero con menor probabilidad)



(*)El proceso aleatorio ayuda a escapar de óptimos locales

(**) A cada individuo se le asocia una parte proporcional a la ruleta según su función de fitness

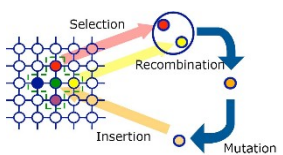


Componentes. Mecanismos de selección.

- Selección de supervivientes:
  - Basada en el fitness: solo sobreviven **los mejores**
  - Basada en la edad: **los descendientes** sustituyen a los padres
  - Combinación (**elitismo**): los mejores individuos siempre sobreviven, independientemente de la edad.
  - Diferentes combinaciones dan lugar a diferentes algoritmos.

- **Günter Rudolph** demostró que **es necesario el elitismo** para que un algoritmo genético pueda converger al optimo global(*)

***(*)Günter Rudolph**: "Convergence Analysis of Canonical Genetic Algorithms". IEEE Transactions on Neural Networks, 5:96-101, January 1994.*



Componentes. Mecanismos de selección.

Otras funciones de selección:

- Selección por Ranking

  Se ordenan los individuos según la función de evaluación y se aplica la función de probabilidad

$$\rho(i) = rank(i)/n * (n-1)$$

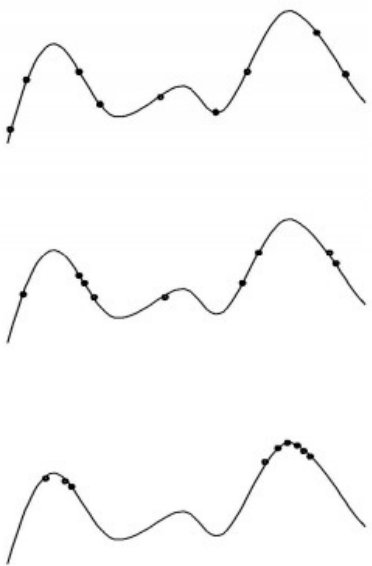
- Selección por torneo

  Se seleccionan k individuos para el “torneo” y se elige al mejor.

Evaluating the impact of different types of crossover and selection methods on the convergence of 0/1 Knapsack using Genetic Algorithm
https://arxiv.org/pdf/2010.03483.pdf

Fases de la ejecución

En una situación ideal(para maximizar):



- **Generaciones iniciales:**

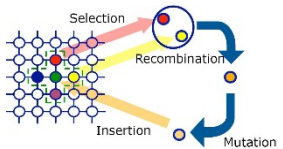
  Distribución aleatoria de la población.

- **Generaciones intermedias:**

  Población cerca de las colinas.

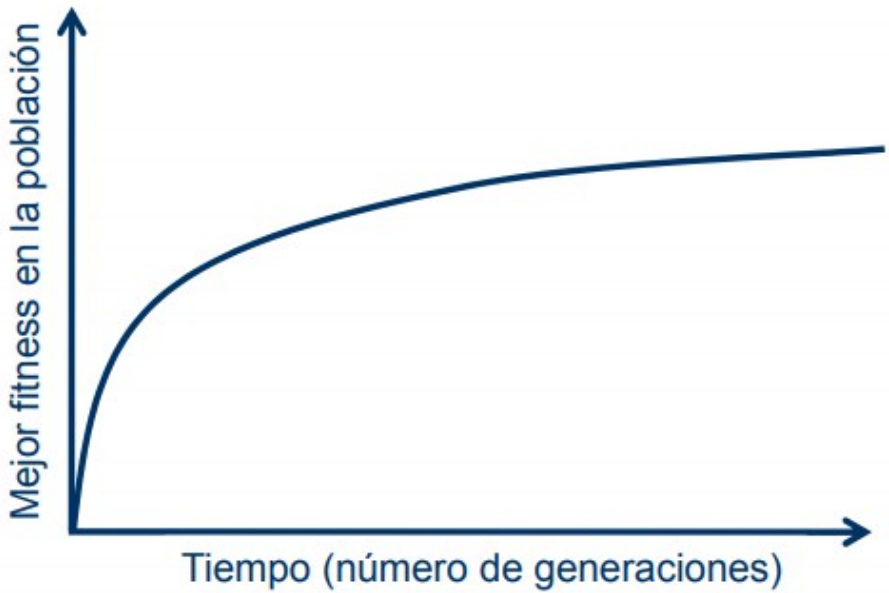
- **Generaciones finales:**

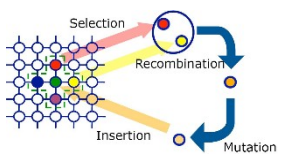
  Población concentrada en las cimas de las colinas más altas.



Fases de la ejecución

Evolución ideal del fitness(para maximizar)

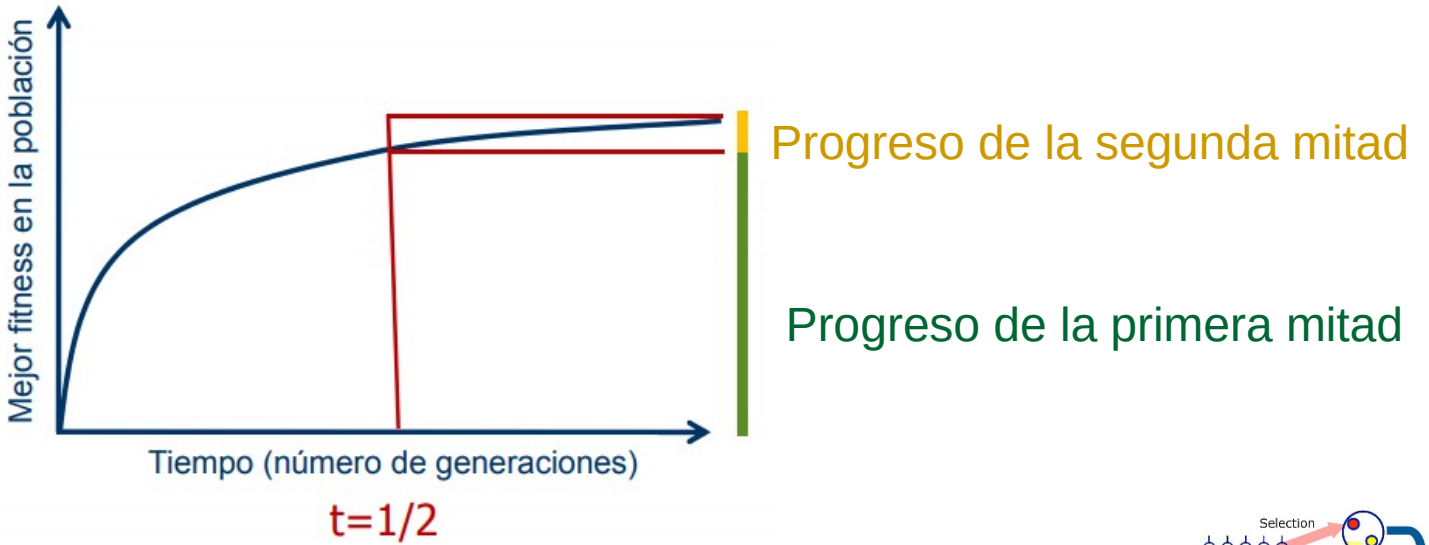


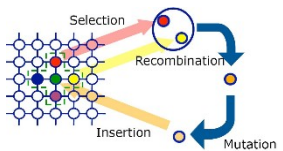


Fases de la ejecución

Evolución ideal del fitness.

¿Merece la pena una ejecución larga (muchas generaciones)?

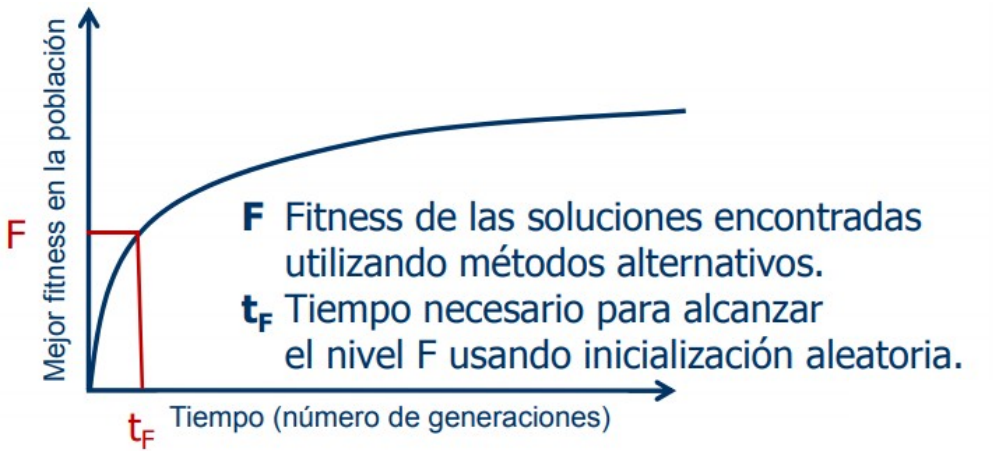


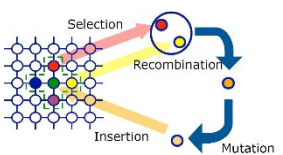


Fases de la ejecución

Evolución ideal del fitness.

¿Merece la pena invertir en la inicialización?(buenas soluciones iniciales)

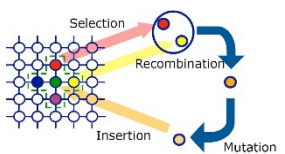




Práctica en Python para resolver el TSP

```python
#Funcion principal del algoritmo genetico
#########################################################3
def algoritmo_genetico(problem=problem,N=100,mutacion=.15,elitismo=.1,generaciones=100):
  # problem = datos del problema
  # N = Tamaño de la población
  # mutacion = probabilidad de una mutación
  # elitismo = porcion de la mejor poblacion a mantener
  # generaciones = nº de generaciones a generar para finalizar

  #Genera la poblacion inicial
  Nodos = list(problem.get_nodes())
  poblacion = generar_poblacion(Nodos,N)

  #Inicializamos valores para la mejor solucion
  (mejor_solucion, mejor_distancia) = Evaluar_Poblacion(poblacion, problem)

  #Condicion de parada
  parar = False
  n=0
  #Inciamos el cliclo de generaciones
  while(parar == False) :

    #Cruce de la poblacion(incluye mutación)
    poblacion = Cruzar(poblacion,mutacion)

    #Seleccionamos la población
    poblacion = Seleccionar(problem,poblacion, N, elitismo)

    #Evaluamos la nueva población
    (mejor_solucion, mejor_distancia) = Evaluar_Poblacion(poblacion, problem)

    print("Generacion #", n, "\nLa mejor solución es:" , mejor_solucion, "\ncon distancia " , mejor_distancia, "\n")

    #Numero de generaciones. Criterio de parada
    if n==generaciones:
      parar = True
    n +=1

  return mejor_solucion
```

https://colab.research.google.com/drive/1o7lG_sJTnVucxryPYJvbRz1b6hJQbwAi



Algoritmo genético para el TSP

Problema del agente viajero(TSP). Datos del problema

```
Total Nodos:42
Total Soluciones:3345252661316380710817006205344075166515200000000
```

La mejor solución encontrada para el problema es: **1273**

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/stsp-sol.html

La mejor solución conseguida:
[39, 24, 40, 21, 9, 23, 41, 8, 29, 32, 34, 33, 20, 35, 36, 17, 31, 0, 1, 3, 4, 6, 5, 13, 19, 14, 16, 15, 37, 7, 26, 18, 12, 11, 25, 10, 2, 27, 28, 30, 38, 22]

con distancia(fitness) 1442

Poblacion=2000, mutacion=.4, elitismo=.5, generaciones=250

Un ejemplo de estudio del TSP con Algoritmos Genéticos

Solving Travelling Salesman Problem Using Improved Genetic Algorithm

https://indjst.org/articles/solving-travelling-salesman-problem-using-improved-genetic-algorithm

Indian Journal of Science and Technology, Vol 10(30), DOI: 10.17485/ijst/2017/v10i30/115512, August 2017
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Solving Travelling Salesman Problem Using Improved Genetic Algorithm

Shweta Rana and Saurabh Ranjan Srivastava

Department of Computer Science, Swami Keshvan Institute of Technology, Management and Gramothan, Ramnagaria, Jagatpura, Jaipur – 302017, Rajasthan, India; miss.shweta.rana@gmail.com, srs@skit.ac.in

Abstract

Objectives: Travelling Salesman Problem is a well known NP-Complete problem. It has many application areas in science and engineering. NP-Complete problems are most difficult problems in computer science. **Methods/Statistical Analysis:** These problems are not solvable using tradition algorithms till date. Soft computing techniques such as Genetic Algorithm (GA) can be used to solve such problems. In this paper Travelling Salesman problem has been solved using Genetic Algorithm. The proposed algorithm modifies the traditional genetic algorithm. Proposed algorithm generates some chromosome using greedy approach and adds these chromosomes in initial population. It also proposes a new greedy cross over operator for genetic algorithm. **Findings:** The implementation results of proposed algorithm prove that proposed algorithm performs better as it find out paths of less path length as compared to the optimal path known till date. **Application/Improvements:** This work is helpful in finding optimal or near optimal solutions of TSP. The algorithm can find application in all the relevant areas of science and engineering where TSP is used such as robot arm movement, drilling electronic boards etc.

Keywords: Travelling Salesman Problem, Genetic Algorithm, Crossover, NP-Complete Problems

1. Introduction

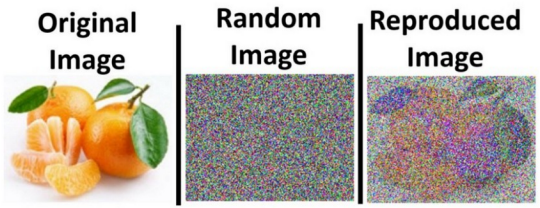
Travelling Salesperson Problem (TSP) is NP-Complete

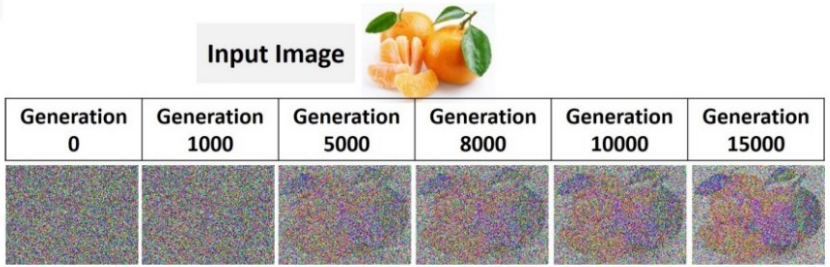
while going to the cities in any request. In asymmetric Travelling salesman Problem the distance between two cities is distinctive while going by the cities in various

Un ejemplo con tratamiento de imagenes

Reproducing Images using a Genetic Algorithm with Python

https://heartbeat.fritz.ai/reproducing-images-using-a-genetic-algorithm-with-python-91fc701ff84





Ampliación de conocimientos y habilidades

Bibliografía

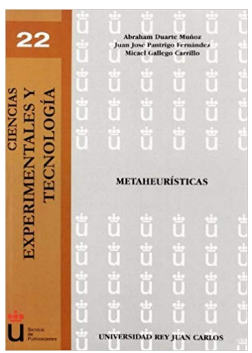
- Duarte, A. (2008). Metaheurísticas. Madrid: Dykinson.

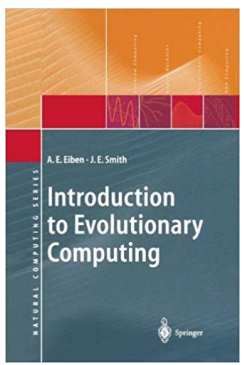
- A.E. Eiben & J.E.Smith(2007). Introduction to Evolutionary Computing

- Carlos Artemio Coello(2014). Introducción a la Computación Evolutiva(Notas de Curso).
http://delta.cs.cinvestav.mx/~ccoello/compevol/apuntes.pdf

Interesante análisis de diferentes representaciones

Practicar





SE2 – Problemas del seminario

03MAIR – Algoritmos de optimización

viu Universidad Internacional de Valencia

Problema 1. Organizar sesiones de doblaje

- Se precisa coordinar el doblaje de una película. Los actores del doblaje deben coincidir en las tomas en las que sus personajes aparecen juntos en las diferentes tomas. Los actores de doblaje cobran todos la misma cantidad por cada día que deben desplazarse hasta el estudio de grabación independientemente del número de tomas que se graben. No es posible grabar más de 6 tomas por día. El objetivo es planificar las sesiones por día de manera que se gasto por los servicios de los actores de doblaje sea el menor posible. Los datos son:

Número de actores: 10
Número de tomas : 30
Actores/Tomas : https://bit.ly/36D8IuK

1 indica que el actor participa en la toma
0 en caso contrario

	Actor									
Toma	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	0	0	0	0	0	
2	0	0	1	1	1	0	0	0	0	0
3	0	1	0	0	1	0	1	0	0	0
4	1	1	0	0	0	0	1	1	0	0
5	0	1	0	1	0	0	0	1	0	0
6	1	1	0	1	1	0	0	0	0	0
7	1	1	0	1	1	0	0	0	0	0
8	1	1	0	0	0	1	0	0	0	0
9	1	1	0	1	0	0	0	0	0	0
10	1	1	0	0	0	1	0	0	1	0
11	1	1	1	0	1	0	0	1	0	0
12	1	1	1	1	0	1	0	0	0	0
13	1	0	0	1	1	0	0	0	0	0
14	1	0	1	0	0	1	0	0	0	0
15	1	1	0	0	0	0	1	0	0	0
16	0	0	0	1	0	0	0	0	0	1
17	1	0	1	0	0	0	0	0	0	0
18	0	0	1	0	0	1	0	0	0	0
19	1	0	1	0	0	0	0	0	0	0
20	1	0	1	1	1	0	0	0	0	0
21	0	0	0	0	0	1	0	1	0	0
22	1	1	1	1	0	0	0	0	0	0
23	1	0	1	0	0	0	0	0	0	0
24	0	0	1	0	0	1	0	0	0	0
25	1	1	0	1	0	0	0	0	0	1
26	1	0	1	0	1	0	0	0	1	0
27	0	0	0	1	1	0	0	0	0	0
28	1	0	0	1	0	0	0	0	0	0
29	1	0	0	0	1	1	0	0	0	0
30	1	0	0	1	0	0	0	0	0	0

Problema 2. Organizar los horarios de partidos de La Liga

- Desde la La Liga de fútbol profesional se pretende organizar los horarios de los partidos de liga de cada jornada. Se conocen algunos datos que nos deben llevar a diseñar un algoritmo que realice la asignación de los partidos a los horarios de forma que **maximice la audiencia**.

- Los horarios disponibles se conocen a priori y son los siguientes:

Viernes	20
Sábado	12,16,18,20
Domingo	12,16,18,20
Lunes	20

Problema 2. Organizar los horarios de partidos de La Liga

- En primer lugar se clasifican los equipos en tres categorías según el numero de seguidores(que tiene relación directa con la audiencia). Hay **3 equipos** en la **categoría A, 11 equipos** de **categoría B** y **6 equipos** de **categoría C.**

- Se conoce estadisticamente la audiencia que genera cada partido según los equipos que se enfrentan y en horario de sábado a las 20h (el mejor en todos los casos)

	Categoría A	Categoría B	Categoría C
Categoría A	2 Millones	1,3 Millones	1 Millones
Categoría B		0.9 Millones	0.75 Millones
Categoría C			0.47 Millones

Problema 2. Organizar los horarios de partidos de La Liga

- Si el horario del partido no se realiza a las 20 horas del sábado se sabe que se reduce según los coeficientes de la siguiente tabla
- Debemos asignar obligatoriamente siempre un partido el viernes y un partido el lunes

	Viernes	Sábado	Domingo	Lunes
12h	-	0.55	0.45	-
16h	-	0.7	0.75	-
18h	-	0.8	0.5	-
20h	0.4	1	1	0.4

Problema 2. Organizar los horarios de partidos de La Liga

- Es posible la coincidencia de horarios pero en este caso la audiencia de cada partido se verá afectada y se estima que se reduce en porcentaje según la siguiente tabla dependiendo del número de coincidencias:

Coincidencias	-%
0	0%
1	25%
2	45%
3	60%
4	70%
5	75%
6	78%
7	80%
8	80%

Problema 2. Organizar los horarios de partidos de La Liga

Los cálculos asociados a una jornada de ejemplo se realizan según se muestra en la siguiente tabla:

Partido	Categorías	Horario	Base(Mill.)	Ponderación	Base*Ponderación	Correción Coincidencia
Celta - Real Madrid	B-A	V20	1,3	0,4	0,52	0,52
Valencia - R. Sociedad	B-A	S12	1,3	0,55	0,72	0,72
Mallorca - Eibar	C-C	S16	0,47	0,7	0,33	0,33
Athletic - Barcelona	B-A	S18	1,3	0,8	1,04	1,04
Leganés - Osasuna	C-C	S20	0,47	1	0,47	0,47
Villarreal - Granada	**B-C**	**D16**	**0,75**	**0,75**	**0,56**	**0,42**
Alavés - Levante	**B-B**	**D16**	**0,9**	**0,75**	**0,68**	**0,51**
Espanyol - Sevilla	B-B	D18	0,9	0,85	0,77	0,77
Betis - Valladolid	B-C	D20	0,75	1	0,75	0,75
Atlético - Getafe	B-B	L20	0,9	0,4	0,36	0,36

Total: 5,88

=0,56 x 0,75

=0,68 x 0,75

Problema 3. Combinar cifras y operaciones

- El problema consiste en **analizar** el siguiente problema y **diseñar** un algoritmo que lo **resuelva**.
- Disponemos de las 9 cifras del 1 al 9 (excluimos el cero) y de los 4 signos básicos de las operaciones fundamentales: suma(+), resta(-), multiplicación(*) y división(/)
- Debemos **combinarlos alternativamente sin repetir ninguno de ellos** para obtener una cantidad dada. Un ejemplo sería para obtener el 4:

  4+2-6/3*1 = 4

Problema 3. Combinar cifras y operaciones

- Debe analizarse el problema para encontrar todos los **valores enteros** posibles planteando las siguientes cuestiones:
  - ¿Qué valor **máximo y mínimo** se pueden obtener según las condiciones del problema?
  - ¿Es posible encontrar t**odos los valores enteros posibles** entre dicho mínimo y máximo ?

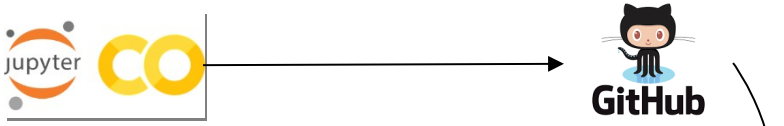
- Nota: Es posible usar la función de python "**`eval`**" para evaluar una expresión:

```
expresion = "4-2+6/3*1"
print(eval(expresion))
```

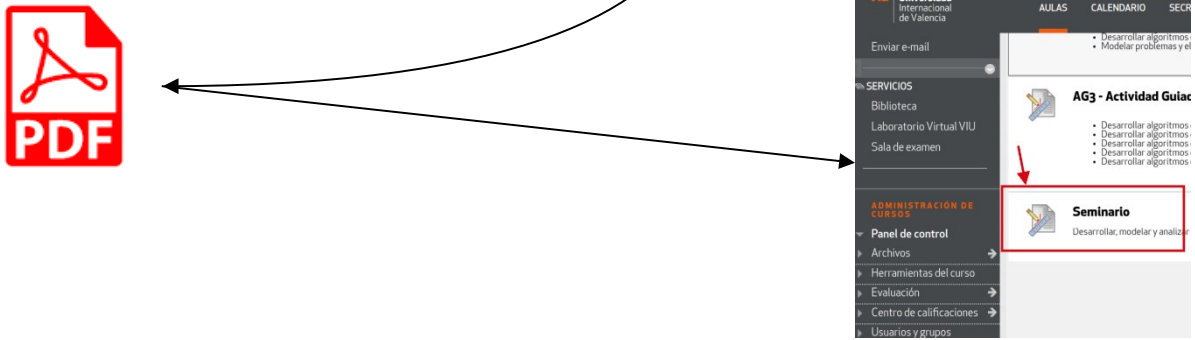
```
4.0
```


Trabajo de Seminario. Entregable

- Generar un Notebook en GitHub (carpeta SEMINARIO)



- Entrega de documento .pdf con en Notebook (como las A. Guiadas)



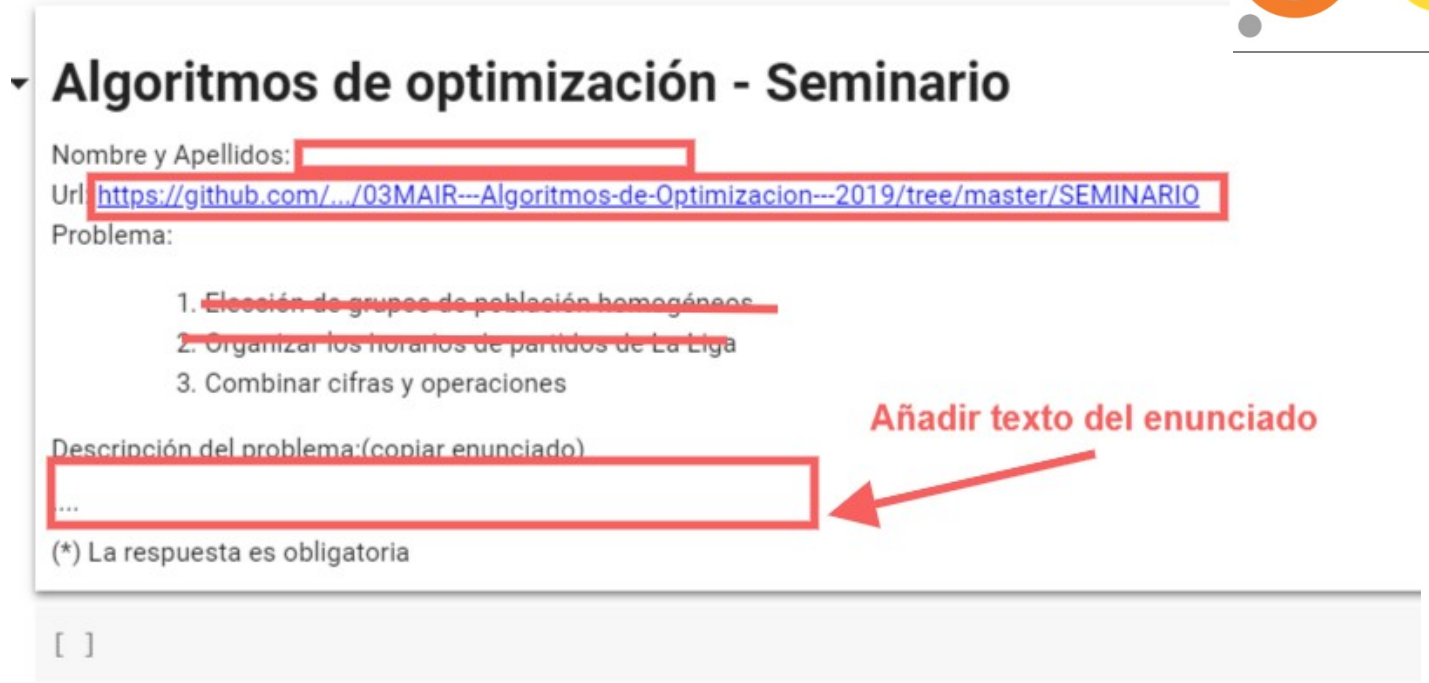
Trabajo de Seminario. Entregable

- Plantilla para el documento

  https://colab.research.google.com/drive/1NVFHsnmrE-wFLX8y1SC3tKlh2et5FOz8

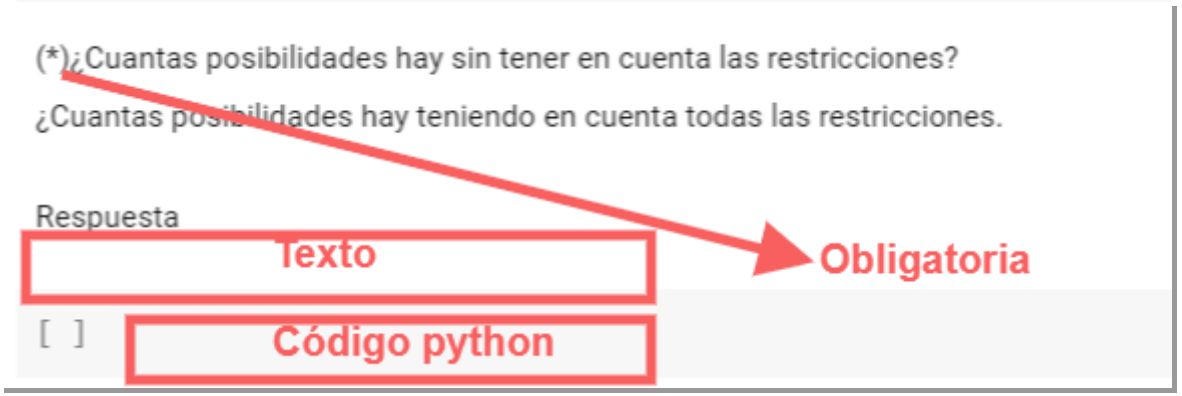
Trabajo de Seminario. Entregable

- Cabecera



Trabajo de Seminario. Entregable

- Pregunta – Respuesta (texto + Python)



Trabajo de Seminario. Entregable. Ejemplo

Pregunta – Respuesta (texto + Python)

(*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta:

Para calcular el numero posible de soluciones es necesario algunas algo de combinatoria y saber contar.

Suponemos que el número de ciudades es N y que partimos de una ciudad dada. Podemos suponer un procedimiento que vaya construyendo todas las soluciones.

Para el primer viaje disponemos de N-1 ciudades ya que debemos eliminar la ciudad de partida como posible candidata. Para la segunda ciudad a visitar disponemos de N-2 posiblidades. Por tanto ya tenemos (N-1)x(N-2) para visitar 2 ciudades.

Si seguimos el razonamiento deducimos que hay (N-1)! (factoria de N-1) posibilidades.

Puesto que el camino es circular(comienza y termina en la misma ciudad) debemos tener en cuenta que cada ruta tiene una ruta inversa semejante. La primera se convierte en la última, la segunda en la penultima y así sucesivamente. Por tanto sin no queremos tener en cuenta esta repeticion en total tenemos (N-1)!/2

Trabajo de Seminario. Preguntas(1/3)

Pregunta – Respuesta (texto + Python)

- **(*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?**
- ¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.
- **(*) ¿Cual es la estructura de datos que mejor se adapta al problema? Argumenta la respuesta**
  (Es posible que hayas elegido una al principio y veas la necesidad de cambiar, argumenta)
- **(*)¿Cual es la función objetivo?**
- **(*)¿Es un problema de maximización o minimización?**

(*) Obligatorio responder

Trabajo de Seminario. Preguntas(2/3)

Pregunta – Respuesta (texto + Python)

- Diseña un algoritmo para resolver el problema por fuerza bruta
- Calcula la complejidad del algoritmo por fuerza bruta
- **(*)Diseña un algoritmo que mejore la complejidad del algoritmo por fuerza bruta. Argumenta porque crees que mejora el algoritmo por fuerza bruta**
- **(*)Calcula la complejidad del algoritmo**
- Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorio.

(*) Obligatorio responder

Trabajo de Seminario. Preguntas(3/3)

Pregunta – Respuesta (texto + Python)

- Aplica el algoritmo al juego de datos aleatorio generado.
- Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo
- Describe brevemente en unas líneas como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posibles variaciones del problema y/o variaciones al alza del tamaño.

Trabajo de Seminario. Evaluación.

Total 13 cuestiones:

6 obligatorias(*) , aseguran 7/10

- **7 opcionales , añaden 2 puntos más: 9/10**
- **1 punto por presentación, descripción**

- lenguaje claro
- código comentado
- acompaña ilustraciones si es necesario(imágenes)

...

Fecha limite de entrega 1ª convocatoria: 01/02/2021

Fecha limite de entrega 2ª convocatoria: 11/02/2021

Actividades Guiadas(*)

- Desarrollar, modelar y analizar algoritmos según diferentes técnicas para resolver el problemas planteados en la asignatura de manera guiada
- Entrega de PDF
- Fecha limite de entrega 1ª convocatoria: 18/01/2021
- Fecha limite de entrega 2ª convocatoria: 11/02/2021

Examen

- Fecha 1ª convocatoria : 18 de enero desde 22:00 (ventana de 48h)
- Fecha 2ª convocatoria : 11 de febrero desde 20:00 (ventana de 48h)
- Duración: 1 hora y 30 minutos
- 10 preguntas: 9 tipo test + 1 práctica de desarrollo

VC6 – Colonia de Hormigas

03MAIR – Algoritmos de optimización

viu Universidad Internacional de Valencia

Metaheurísticas. Método Constructivo. Colonia de Hormigas

Esquema básico

```
Depositar una cantidad de feromona inicial en todas las aristas
Crear m hormigas
Repetir:
    Reiniciar hormigas (borrar memoria)
    Cada hormiga: Construir solución usando feromonas y coste de las aristas
    Cada hormiga: Depositar feromonas en aristas de la solución
    Evaporar feromona en las aristas
Devolver: la mejor solución encontrada
```

Metaheurísticas. Método Constructivo. Colonia de Hormigas

Esquema básico (I) (1ª iteración)

```
def hormigas(problem, N) :
  #problem = datos del problema
  #N = Número de agentes(hormigas)

  #Nodos
  Nodos = list(problem.get_nodes())
  #Aristas
  Aristas = list(problem.get_edges())

  #Inicializa las aristas con una cantidad inicial de feromonas:1
  #Mejora: inicializar con valores diferentes dependiendo diferentes criterios
  T = [[ 1 for _ in range(len(Nodos)) ] for _ in range(len(Nodos))]

  #Se generan los agentes(hormigas) que serán estructuras de caminos desde 0
  Hormiga = [[0] for _ in range(N)]
```

Propuesta de mejora

Se inicializa todo con cero en el primer nodo

continua

Metaheurísticas. Método Constructivo. Colonia de Hormigas

Esquema básico (II) (1ª iteración)

```
#Recorre cada agente construyendo la solución
for h in range(N) :
  #Para cada agente se construye un camino
  for i in range(len(Nodos)-1) :

    #Elige el siguiente nodo
    Nuevo_Nodo = Add_Nodo(problem, Hormiga[h] ,T )
    Hormiga[h].append(Nuevo_Nodo)

  #Incrementa feromonas en esa arista
  T = Incrementa_Feromona(problem, T, Hormiga[h] )
  #print("Feromonas(1)", T)

  #Evapora Feromonas
  T = Evaporar_Feromonas(T)
  #print("Feromonas(2)", T)

  #Seleccionamos el mejor agente
mejor_solucion = []
mejor_distancia = 10e100
for h in range(N) :
  distancia_actual = distancia_total(Hormiga[h], problem)
  if distancia_actual < mejor_distancia:
    mejor_solucion = Hormiga[h]
    mejor_distancia =distancia_actual
```

N=Nº de hormigas

Añade un nuevo nodo

T=Lista de aristas

Metaheurísticas. Método Constructivo. Colonia de Hormigas

Funciones auxiliares

```
def Add_Nodo(problem, H ,T ) :
  #Mejora:Establecer una funcion de probabilidad para
  # añadir un nuevo nodo dependiendo de los nodos mas cercanos y de las feromonas depositadas
  Nodos = list(problem.get_nodes())
  return random.choice(   list(set(range(1,len(Nodos))) - set(H) )  )
```

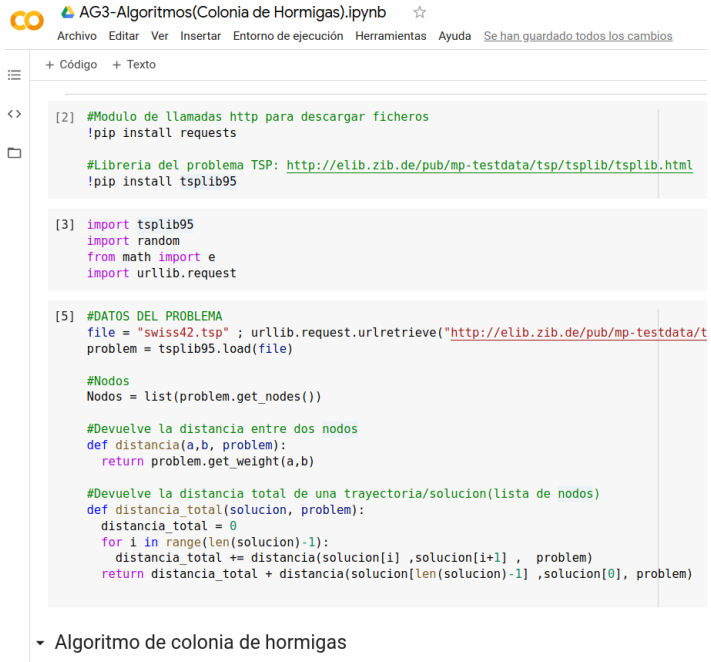
Poco eficiente, demasiado aleatoria

```
def Incrementa_Feromona(problem, T, H ) :
  #Incrementa segun la calidad de la solución. Añadir una cantidad inversamente proporcional a la distancia total
  for i in range(len(H)-1):
    T[H[i]][H[i+1]] += 1000/distancia_total(H, problem)
  return T

def Evaporar_Feromonas(T ):
  #Evapora 0.3 el valor de la feromona, sin que baje de 1
  #Mejora:Podemos elegir diferentes funciones de evaporación dependiendo de la cantidad actual y de la suma total de feromonas depositadas,..
  T = [[ max(T[i][j] - 0.3 , 1) for i in range(len(Nodos)) ] for j in range(len(Nodos))]
  return T
```

Metaheurísticas. Método Constructivo. Colonia de Hormigas

Trabajar sobre

https://colab.research.google.com/drive/1nX4FZJGCCmh31ps8znXMSxbdgimGW0T2?usp=sharing



Metaheurísticas. Método Constructivo. Colonia de Hormigas

Función de probabilidad para elegir el siguiente nodo

Cantidad de feromonas en la arista (i,j)

Inversa de la distancia de i a j

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\nu_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha [\nu_{il}]^\beta}, \text{ si } j \in J_i^k$$

$$p_{ij}^k(t) = 0, \text{ si } j \notin J_i^k$$ → No se elije si no está en los nodos pendientes

α	Peso relativo que se da a la elección por el rastro
β	Peso relativo que se da a la elección por la distancia entre dos nodos

Si α es igual a 0 , se asemeja a una técnica voraz
Si β es igual a 0 , solo intervine la feromona que puede no ser lo ideal

Metaheurísticas. Método Constructivo. Colonia de Hormigas

10/10

Lineas de mejora para:

- Iterar, generar nuevas hormigas con la información de las feromonas
- Función de probabilidad para elegir el siguiente nodo (parámetros α β

Cantidad de feromonas en la arista (i,j)

Inversa de la distancia de j a j

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\nu_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha [\nu_{il}]^\beta}, \text{ si } j \in J_i^k$$

$$p_{ij}^k(t) = 0, \text{ si } j \notin J_i^k$$ → No se elije si no está en los nodos pendientes

α	Peso relativo que se da a la elección por el rastro
β	Peso relativo que se da a la elección por la distancia entre dos nodos

Si α es igual a 0 , se asemeja a una técnica voraz
Si β es igual a 0 , solo intervine la feromona que puede no ser lo ideal

Metaheurísticas. Método Constructivo. Colonia de Hormigas

Algoritmo ACO aplicado al TSP: Resumen de una experiencia práctica

Benjamín Arenas F., benarenas@hotmail.com
Alejandro Pavez S., apavez@mi.terra.cl
Rodrigo Vidal K., rovika@yahoo.com
Universidad Técnica Federico Santa María
Departamento de Informática

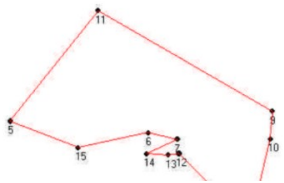
Abstract: La metaheurística Ant Colony Optimization (ACO) [4] es uno de los nuevos paradigmas de resolución de problemas combinatorios del tipo NP. En este artículo se presentan los resultados obtenidos al aplicar un algoritmo variante de la metaheurística ACO en torno al mundialmente conocido Traveling Salesman Problem (TSP) [10], se discuten las oportunidades de desarrollo futuro y se presentan nuevos tuors solución con mejores resultados que los mundialmente encontrados hasta hoy.

Palabras claves: TSP, ACO, Simulated Annealing, Metaheurística

1. Introducción

La importancia del TSP dentro del mundo de problemas NP radica en que es utilizado como conjunto de pruebas para los nuevos algoritmos asociados a la resolución de esta clase de problemas. Su naturaleza NP-Completo hace posible que se puedan resolver una gran familia de problemas equivalentes mediante transformaciones particulares para cada caso [8]. Además, como el TSP presenta una gran cantidad de aplicaciones en el mundo real, se ha convertido en uno de los problemas más estudiados por la comunidad científica mundial. Las características de la nueva metaheurística ACO, aplicadas al TSP [2], [3], [7], nos puede ayudar a encontrar soluciones

dos veces una misma ciudad. El objetivo es minimizar la longitud de la secuencia de visita de las ciudades, bajo el criterio de minimización (maximización) de la función objetivo. Un ejemplo de un tour es el que se muestra a continuación en el figura 1.



https://www.academia.edu/6676146/Algoritmo_ACO_aplicado_al_TSP_Resumen_de_una_experiencia_pr%C3%A1ctica

Resumen de lo aprendido

- Conocer las técnicas clásicas deterministas:

- Voracidad, Divide y vencerás, Programación dinámica, Programación lineal
- Búsquedas con grafos

- Conocer las implicaciones de la Complejidad computacional

- Conocer problemas tipo para clasificar(si se puede) nuestros próximos problemas a resolver

- Tener destreza para modelizar:

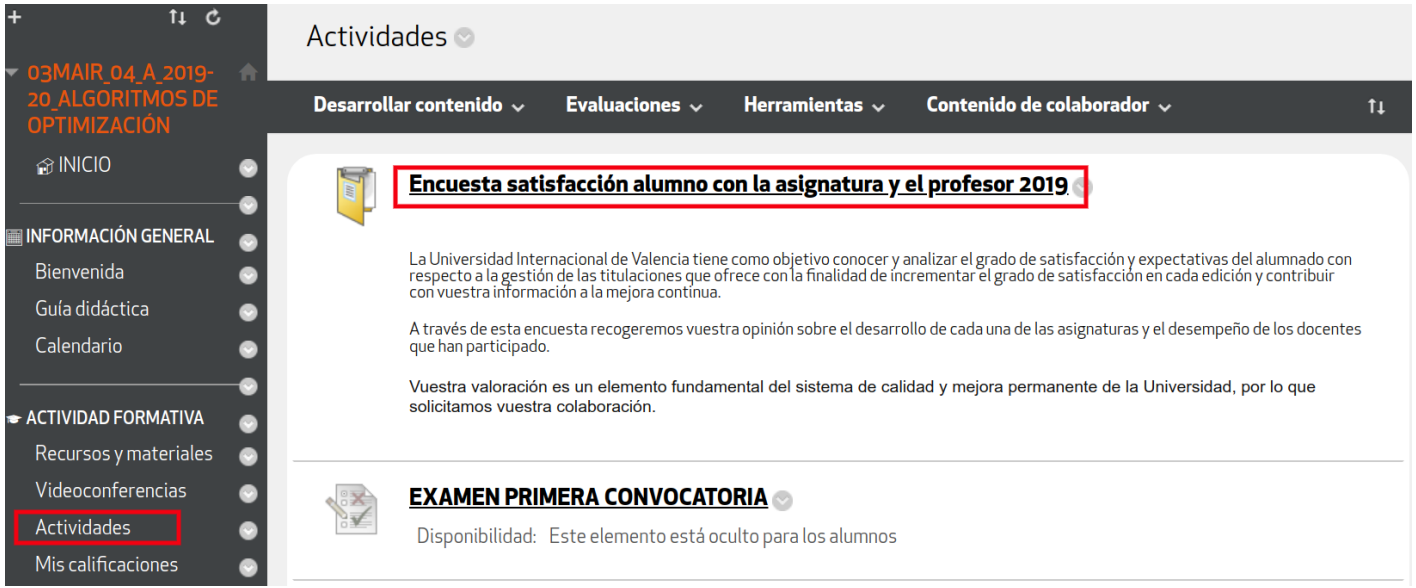
- Espacio de soluciones, función objetivo, restricciones

- Conocer técnicas metaheurísticas:

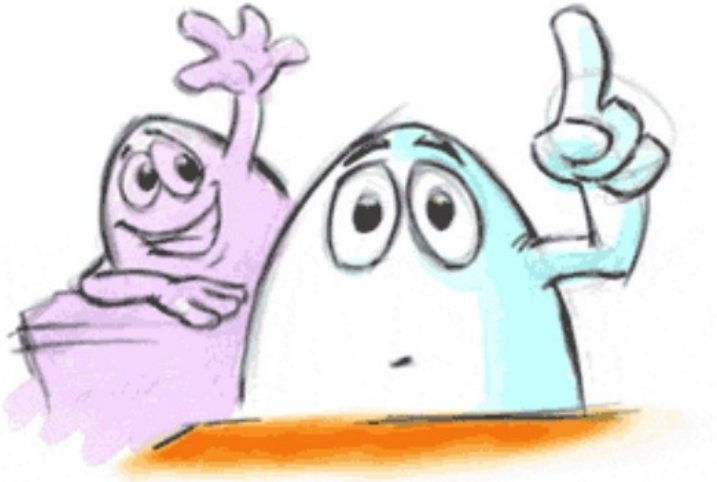
- Trayectorias, constructivas, poblacionales,…

- Practicar → Investigar →Practicar → Investigar →Practicar → Investigar ...

Actividad. Encuesta de satisfacción



¿Preguntas?



Gracias

raul.reyero@campusviu.es

viu Universidad Internacional de Valencia