



# Algoritmo de escalada

## Ejercicio 1

El algoritmo de escalada no encuentra el óptimo global, encuentra el local y en cada una encuentra soluciones distintas porque hay una variable aleatoria en el código para la ciudad inicial y la final. En la eficiencia y optimalidad por una parte el tiempo de ejecución es muy aceptable a la hora de encontrar la solución pero la calidad de ésta decrece debido a que puede quedarse estancado en una meseta además de no encontrar el óptimo global, en esto guarda semejanza con el algoritmo en profundidad, con propiedades similares, salvo por la diferencia de que la escalada se guía por una heurística

# Enfriamiento simulado

## Ejercicio 2

Tras ejecutar los ejemplos vemos que todas las veces ha encontrado la solución óptima global, viendo la lista de soluciones comprobamos que los resultados no tienen valores crecientes si no que van variando, esto es que al escoger de manera aleatoria el próximo movimiento que se va a utilizar puede escoger un vecino peor que más tarde le lleva a un resultado mejor.

Al ejecutar los ejemplos con el algoritmo de máxima pendiente vemos que ninguno de los resultados es el óptimo, al tener valores repetidos y no muy diferentes, tiende a quedarse en óptimos locales al descartar cualquier opción que sea peor que la que tiene.

# Algoritmos genéticos

## Ejercicio 3

Para definir la población hemos creado una lista donde vamos a introducir en cada vuelta el individuo que estamos creando, para crear cada individuo, le pasamos a la función random.choices los genes de nuestro problema y la longitud de los individuos.

Para realizar el cruce obtenemos de dos en dos los individuos que queremos cruzar, sacamos su longitud y para el primer hijo cogemos la primera mitad del primer padre y la segunda mitad del segundo padre, para el segundo hijo cogemos la segunda mitad del primer padre y la primera mitad del segundo padre. Cada hijo generado lo introducimos en una lista que es la que devolvemos.

La población generada de una mutación de individuos la obtenemos llamando a la función de mutación para cada miembro de la población y metiendo en una lista el resultado, esta lista es nuestra nueva población

## Ejercicio 4

En la selección por torneo, para seleccionar cada nuevo individuo, se selecciona k individuos de la población inicial y los metemos en la función max o min, teniendo en cuenta la función de fitness de cada uno de ellos.

## Ejercicio 5

Al realizar la resolución del problema usando algoritmos genéticos usando la función de min, el resultado es (0,0), esto es porque el valor más pequeño que se puede obtener es el cero y su función de fitness es 0, que es el valor elevado al cuadrado. Este resultado se obtiene de manera fácil ya que cada vez que se hace un torneo se obtiene el valor más bajo, en cada generación nos quedamos con los resultado más bajos, obteniendo finalmente el 0. Para el caso en la que la función que queremos es maximización el resultado es (1023, 1046529) que es el valor más alto que se puede obtener con 10 bits y su cuadrado, que es su función de fitness, en cada generación, en el torneo nos quedamos con los valores más altos, esto hace que nos acerquemos cada vez al valor más alto posible.

Si el número de generaciones es muy bajo y se cogen menos individuos para el torneo, es más difícil que se llegue a una buena solución ya que al realizar las mutaciones y los cruces de manera aleatoria no tiene que por qué generar muy buenos individuos de manera rápida.

# Problema de la mochila

## Ejercicio 6

En este ejercicio probamos 2 iteraciones distintas, la primera consiste en calcular la diferencia entre el peso y el valor de cada objeto, con lo que esperábamos encontrar los objetos con la mejor relación entre su bajo peso y su valor, y al principio parecía funcionar, en el ejercicio con 4 objetos propuestos devolvía el mejor, y en el de 10 conseguía encontrar la suma correcta, que al analizarlo esa suma sólo era posible con el gen óptimo, sin embargo al continuar con la exploración el algoritmo no encontró la solución óptima para 15 objetos en adelante. Descubrimos que en el caso de encontrar un objeto pesado de gran valor, y dos cuyo peso era igual o inferior pero su suma mayor, escogía el pesado (Capacidad 4 , P[5,3,3] , V[7,4,4]).

La segunda consiste en crear una matriz con la suma del valor acumulado si entra dentro del peso posible, esta solución dependía de ordenar con anterioridad el vector de pesos, para lo que recurrimos a un quickSort modificado para que cambiase de posición al vector valores imitando al vector pesos, tras lo cual crea la matriz y la rellena, siendo el valor buscado el último de la matriz V[N][P] (N: Número de elementos, P: Capacidad máxima).

## Ejercicio 7

Al ejecutar las distintas instancias del problema Problema\_Genetico vimos cómo al aumentar la cantidad de elementos a procesar el tiempo se disparaba, debido al coste del algoritmo propuesto, cercano a ser cuadrático, el tiempo era comprensible para los 2 primeras variables m1g y m2g con el primer algoritmo, sin embargo empezaba a tardar varios minutos con la primera iteración de m3g, los restantes al ser de un orden de magnitud superior en número de elementos es esperable que su tiempo de resolución también lo sea. El algoritmo no encuentra la mejor solución cuando se encuentra con uno de los casos expuestos en el ejercicio 6.

El segundo algoritmo en cambio siempre encuentra la solución óptima, pero su tiempo de ejecución es mucho más amplio que el del primer algoritmo, haciendo casi inviable probarlo con los casos con mayor número de elementos, o mayor capacidad.