Práctica 5 – Boletín de Ejercicios

Pedro López-Chaves Pérez

Búsqueda tabú

Ejercicio 1.1

En el operador de inserción, lo que sucede es que se introduce cada ciudad en cualquier posición, por ello, el primer elemento se puede insertar en todos los puestos menos el suyo, el siguiente en todos menos el suyo y el intercambio con el anterior y así sucesivamente. Es por ello que la fórmula que recoge el tamaño del vecindario es $N(S0) = \sum_{i=0}^{N-1} N - i$.

En el caso de inversión simple, sería obtener todos los pares de índices, sin repetir combinaciones. El primero se combinará con todos menos él, el segundo con todos menos él y el primero (ya que esa combinación ya existe), y así sucesivamente. Es por ello que la fórmula que recoge el tamaño del vecindario es $N(S0) = \frac{Nx(N-1)}{2}$.

El tamaño total de soluciones es la permutación de todos los elementos, siendo por ello N!. Es por ello que el porcentaje del espacio de soluciones que abarca cada solución se calcularía de la siguiente manera $\frac{N(S0)}{N!}x100$; siendo N(S0) las mencionadas anteriormente en cada caso.

Cabe destacar que N(S0) en el algoritmo real estaría condicionado por la lista tabú, ya que las combinaciones que se encontrasen en la misma se eliminarían del vecindario también.

Ejercicio 1.2

A) Se eliminarían los intercambios entre índices que se recogen en la lista tabú que son 0,3 y 0,1. Los eliminados del vecindario serían 2,1,3,4 y 4,2,3,1; y nos quedarían 1,4,3,2; 1,2,4,3;2,1,3,4;1,3,2,4.
B) Hay elementos que no serían compatibles con una inserción, ya que solo se mueve un número, los que no valen son: 1,4,3,2; 3,2,1,4; 4,2,3,1. Por otro lado, descartaríamos los que nos prohíbe la lista tabú: 2,1,3,4. Nos quedaría entonces: 1,3,2,4 y 1,2,4,3.
C) No nos sirven las soluciones donde los valores de las posiciones en las que en el índice 1 esté el valor 2, ni en el índice 1 esté el valor 4. Nos quedaría el siguiente vencindario: 1,3,2,4; 2,1,3,4; 4,2,3,1.

Enfriamiento Simulado

Ejercicio 1.4

A) Al ser una solución peor que la actual, se aceptaría si la probabilidad fuese igual o menor que 97%, y al no ser este el caso, no se acepta.
B) Se acepta ya que el coste es menor que el actual.
C) El coste es superior, por ello no se acepta, pero sí se acepta ya que la probabilidad es inferior al 97%.
D) No se permite ya que sobrepasa el límite de 1200 fijado.

Ejercicio 1.5

Si la temperatura es igual a 0 durante toda la ejecución, nunca se van a aceptar soluciones que tengan un coste mayor del que ya hemos conseguido y hará que nos quedemos en mínimos locales, al no poder explorar más allá de estos.

En cambio, si la temperatura en infinita, aceptaremos cualquier solución sin importar su coste. Esto dará lugar a que obtengamos soluciones totalmente aleatorias.

Algoritmos genéticos

Ejercicio 1.8

- Partially Mapped Crossover: para este método, cogemos los 2 puntos seleccionados, y a partir de ahí creamos los hijos añadiendo unas dependencias. Primero obtenemos los hijos sin las correspondencias:
  - H1: (6, 9, 3, 9, 5, 1, 3, 1, 2)
  - H2: (7, 2, 7, 8, 5, 4, 6, 8, 4)

  Ahora aplicamos las correspondencias que son 3-7, 9-8, 5-5,1-4-3:

  - H1: (6, 8, 3, 9, 5, 1, 7, 4, 2)
  - H2: (3, 2, 7, 8, 5, 4, 6, 9, 1)

- Order Crossover: se coje el tramo seleccionado, se rellena con el otro padre en orden desde justo después los puntos seleccionados:
  - H1: (8, 3, 7, 6, 5, 4, 1, 9, 2)
  - H2: (7, 6, 2, 8, 3, 4, 2, 1, 9)

- Cycle Crossover: aplicamos, seleccionando de inicio las primeras posiciones los primeros ciclos:
  - H1: (8, *, 6, 1, *, *, *, 9, *)
  - H2: (9, *, 8, 6, *, *, *, 1, *)

  Aplicamos ahora el segundo ciclo a cada uno, escogiendo tambien la primera posición disponible para cada uno:

  - H1: (8, 3, 6, 1, *, 2, *, 9, 4)
  - H2: (9, 4, 8, 6, *, 3, *, 1, 2)

  Por último, aplicamos el tercer y último ciclo:

  - H1: (8, 3, 6, 1, 7, 2, 5, 9, 4)
  - H2: (9, 4, 8, 6, 5, 3, 7, 1, 2)

Ejercicio 1.10

Algoritmo Genético para el Problema de las 8 Reinas:

a) Representación de la solución y tamaño del espacio de búsqueda:

Para la representación de la solución se presentará un vector de enteros de tamaño 8. Cada valor del vector representará la posición de cada reina en la fila, ya que, hay 8 filas y cada reina ha de estar en una diferente.

El tamaño del espacio de búsqueda será de 8 elevado a 8, ya que hay 8 filas y en cada fila 8 posibles soluciones

b) Función de fitness:

La función fitness devolvería el número de reinas en conflicto, que se estuviesen amenazando y sería 0 al alcanzar la solución final.

c) Operadores de selección, cruce y mutación:

Para la selección, se puede utilizar la selección por torneo para seleccionar individuos prometedores.

El operador de cruce puede ser el cruce el cruce de 2 puntos, a partir de los vectores de 2 ancestros.

Mutación: Se puede aplicar una mutación desplazando una reina a otra posición de su fila aleatoriamente.

d) Mecanismo de reemplazo poblacional:

Podría ser un reemplazo que mantenga la mitad más prometedora de los ancestros y la mitad mas prometedora de los descendientes, lo que sería elitismo.

Algoritmo Genético para el Problema de la Mochila:

a) Representación de la solución y tamaño del espacio de búsqueda:

La solución se representa como un vector binario de longitud n, donde n es el número de elementos en la mochila. Un 1 indica que el elemento está en la mochila, y 0 indica que no lo está, es decir, la decisión que se toma.

El tamaño de la búsqueda es de 2 elevado a n, ya que, cada elemento tiene dos posibles estados (en la mochila o no).

b) Función de fitness:

La función de fitness sería una resta de el valor de la capacidad de la mochila menos la suma de los valores de los objetos seleccionados iría decreciendo cuanto más nos acercásemos a llenar la mochila y si nos pasásemos del tamaño, los valores negativos los tomaríamos como no válidos.

c) Operadores de selección, cruce y mutación:

Para la selección se puede usar selección por torneo para seleccionar soluciones para la reproducción.

Para el cruce usaremos el cycle crossover con los valores de la decisión.

La mutación se puede aplicar mutación invirtiendo el valor de bits con una cierta probabilidad.

d) Mecanismo de reemplazo poblacional:

Utilizaremos de nuevo el elitismo.

Programación genética

Ejercicio 1.11

A) Las expresiones serían las siguientes: (x+1)-0, 1+(x*x), 2+0, x*((-1)-(-2)).
B) Para la realización de estas pruebas creamos un script de python, con el que obtenemos el error absoluto para los valores indicados. Observamos como el error va en ascenso:

```
In [6]: n=[-1, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2

a=0
b=0
c=0
d=0

for i in n:
    f_obj = i*i + i+ 1.0
    #vamos calculando el error absoluto para cada valor y lo acumulamos
    a += abs(f_obj - (i+1)-0)
    b += abs(f_obj - 1 + (i*i))
    c += abs(f_obj - 2+0)
    d += abs(f_obj - (i*(-1 - -2)))

print("A) Error", round(a,2))
print("B) Error", round(b,2))
print("C) Error", round(c,2))
print("D) Error", round(d,2))
```

```
A) Error 7.7
B) Error 16.2
C) Error 17.98
D) Error 28.7
```

C) Las operaciones con punto de corte se convierten en x-0 y 1+(x+1)*x, ejecutamos el script y volvemos a obtener los resultados:

```
n=[-1, -0.9, -0.8, -0.7, -0.6, -0.5, -0

a=0
b=0

for i in n:
    f_obj = i*i + i+ 1.0
    a += abs(f_obj - i-0)
    b += abs(f_obj - (1 + ((i+1)*i)))

print("A) Error", round(a,2))
print("B) Error", round(b,2))
```

```
A) Error 28.7
B) Error 0.0
```

Se observa como b es la misma operación que f_obj, por lo que el error es 0.

D) La operación C queda en nada y D queda en x*2, obtenemos de nuevo los resultados:

```
n=[-1, -0.9, -0.8, -0.7, -0.6, -

d=0

for i in n:
    f_obj = i*i + i+ 1.0
    d += abs(f_obj - i*2)

print("D) Error", round(d,2))
```

```
D) Error 28.7
```

Algoritmos de optimización de colonia de hormigas

Ejercicio 1.12

La regla de actualización local consiste en que cada hormiga deposita y evapora una cantidad de feromona en cada arista que visita durante su recorrido. Por otro lado, la regla de actualización global se aplica después de que todas las hormigas hayan completado su recorrido y consiste en actualizar (evaporar y depositar) la feromona existente en todas las aristas del grafo.

Si la ratio de evaporación de feromona fuera alta, el efecto esperado sería una disminución más rápida de la cantidad de feromona en las aristas. En el caso de la regla de actualización local, esto significaría que las soluciones óptimas encontradas por las hormigas tendrían una vida útil más corta, ya que la feromona depositada se evaporaría rápidamente. Esto podría llevar a una mayor exploración del espacio de soluciones por parte de las hormigas y a una mayor diversificación de las soluciones encontradas. En el caso de la actualización global sería

parecido, diversificaría la exploración, investigando poco las soluciones obtenidas en cada una de estas diversificaciones, al disminuir muy rápido las feromonas.

Ejercicio 1.13

En el sistema de hormigas (AS), las hormigas también utilizan una regla de decisión probabilística similar para seleccionar la siguiente ciudad, pero en este caso, consideran tanto la feromona como una heurística que proporciona información adicional sobre la calidad de la ciudad. Normalmente esta heurística, se establece al inicio, siendo el inverso de las distancias.

En el sistema de hormigas elitista (EAS), cada hormiga construye su solución utilizando una regla de decisión probabilística simple, donde selecciona la siguiente ciudad para visitar basándose únicamente en la feromona depositada por las hormigas anteriores. La cantidad de feromona almacenada estará relacionada con lo buena que sea la solución, es decir, cuanto menor sea el coste.

En el algoritmo de sistema de colonias de hormigas (ACS), se introducen dos mejoras principales. Se añade la actualización global y local de las feromonas; en la búsqueda de un equilibrio entre guiar la búsqueda hacia las soluciones más prometedoras y diversificar la búsqueda evitando caer en mínimos locales.

En resumen, en los 2 primeros hay exploración y el tercero hay exploración y explotación.

Si establecemos el parámetro de importancia relativa de la feromona en cero (0), la influencia de la feromona en la decisión de las hormigas se anula y solo se tiene en cuenta la heurística. De esta manera, se obtiene una regla de decisión equivalente a la del sistema de hormigas (AS), donde solo se considera la información heurística para seleccionar la siguiente ciudad a visitar.

Algoritmos de enjambre de partículas

Ejercicio 1.15

Para este ejercicio, las fórmulas para el ajuste de la velocidad son las siguientes (siendo rnd igual a 0.5):

$$v_{id} = v_{id} + \underbrace{\varphi_1 \cdot rnd() \cdot (pBest_{id} - x_{id})}_{\text{COGNITIVO}} + \underbrace{\varphi_2 \cdot rnd() \cdot (g_{id} - x_{id})}_{\text{SOCIAL}}$$
$$x_{id} = x_{id} + v_{id}$$

Rellenando ahora los campos, realizamos los cálculos para cada caso:

V1= 0.1*(2,2)+2*rnd*((5,5)-(5,5))+2*rnd*((5,5)-(5,5))=(0.2, 0.2)

X1=(5,5)+(0.2, 0.2)= (4.8, 4.8)

V2=0.1*(3,3)+2*rnd*((7,3)-(8,3))+2*rnd*((5,5)-(8,3))=(-4 , 2.5)

X2=(8,3)+ (-4 , 2.5)=(4, 5.5)

V3=0.1*(4,4)+2*rnd*((5,6)-(6,7))+2*rnd*((5,5)-(6,7))=(-1.1 , -2.1)

X3=(6,7)+(-1.1, -2.1)=(4.9, 4.9)

La inercia lo que hace es determinar el peso que tendrá la velocidad actual en la nueva. Si el valor de la inercia es alto, se traduce en que la inercia sea prácticamente nula; pero ponerlo bajo, es como suprimir el peso de la velocidad actual, permitiendo aceleraciones gigantescas.

Ejercicio 1.16

A) La diferencia es que las poblaciones de los algoritmos genéticos se combinan en pares o diferentes combinaciones aleatoriamente para generar nuevos individuos; mientras que en los algoritmos de enjambre de partículas hay tanto un factor social y local para la generación de nuevos individuos.
B) La diferencia es que los algoritmos de colonia de hormigas es que los algoritmos de enjambre de partículas se comportan de manera social, es decir, se busca mejorar el entorno en general; mientras que, en los de colonia de hormigas, los individuos se comportan de manera individual e independiente, hasta logras alguna solución.

Búsqueda con adversario

Ejercicio 2.1

Nos encontramos intentando solucionar el 3 en raya mediante búsqueda con adversario. Para ello se calcula una función de utilidad en la que calculamos 2 valores: las filas, columnas y diagonales con las que A puede ganar y las filas, columnas y diagonales con las que B puede ganar. Dependiendo de quien sea el turno, utilizaremos una decisión de Max o de Min, ante la resta de estos valores.

- 4-2=2: A puede ganar con la diagonal, la fila inferior o las 2 columnas de la derecha; B solo puede ganar con la fila superior y con la columna de la izquierda.
- 4-2=2: A puede ganar con las 2 diagonales, la columna de la derecha y la fila inferior; B puede ganar con la fila superior y la columna de la izquierda.
- 3-2=1: A puede ganar con una diagonal, la fila inferior y la columna central; B puede ganar con la fila superior y la columna izquierda.
- 5-2=3: A puede ganar con las 2 diagonales, las filas inferior y superior, y la columna central; B puede ganar con la columna de la izquierda y la fila superior.
- 4-2=2: A puede ganar con la columna derecha, las 2 diagonales y la fila superior; B puede ganar con la columna de la izquierda y la fila superior.
- 4-2=2: A puede ganar con la diagonal, las columnas de la derecha y la fila superior; B puede ganar con la columna de la izquierda y la fila superior.

Ejercicio 2.2

Se poda el subárbol central y esto se debe a la estructura min max del árbol. Primero, tenemos una selección por min y luego otra por max. El subárbol anterior nos devolvió el valor 3, al ser este el menor del subárbol. A continuación, seleccionamos el primer elemento del subárbol central que es 2, como es una decisión por min, los valores que obtengamos en este subárbol solo podrán ir de 2 a menos infinito. El problema es que el primer subárbol nos devolvió un 3, entonces, no hay manera posible que la siguiente decisión, que se basa en max, devuelva un valor de este subárbol, por ello lo podamos.

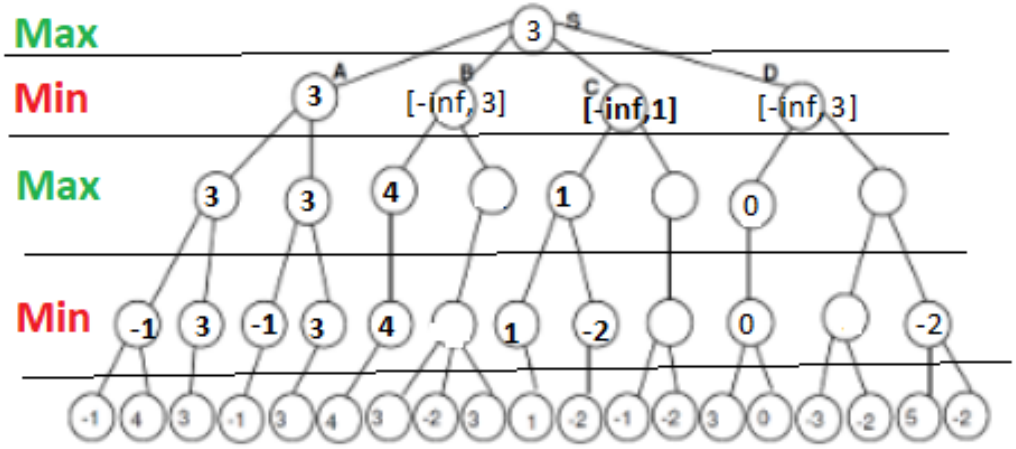
Ejercicio 2.3

Para este ejercicio, primero marcamos los niveles de max y de min. A continuación empezamos la algoritmia y podado a la vez. En el subárbol A obtenemos como valor 3, es decir, cualquier subárbol que no pueda obtener un valor mayor a este lo podaremos.

En el subárbol B, podemos observar cómo obtenemo 4, pero en el nodo anterior se elige el menor y el mayor de los iniciales de la otra rama es 3, en el mejor de los casos se elegiría este que es el mismo valor que ya tenemos, pues ya lo podamos.

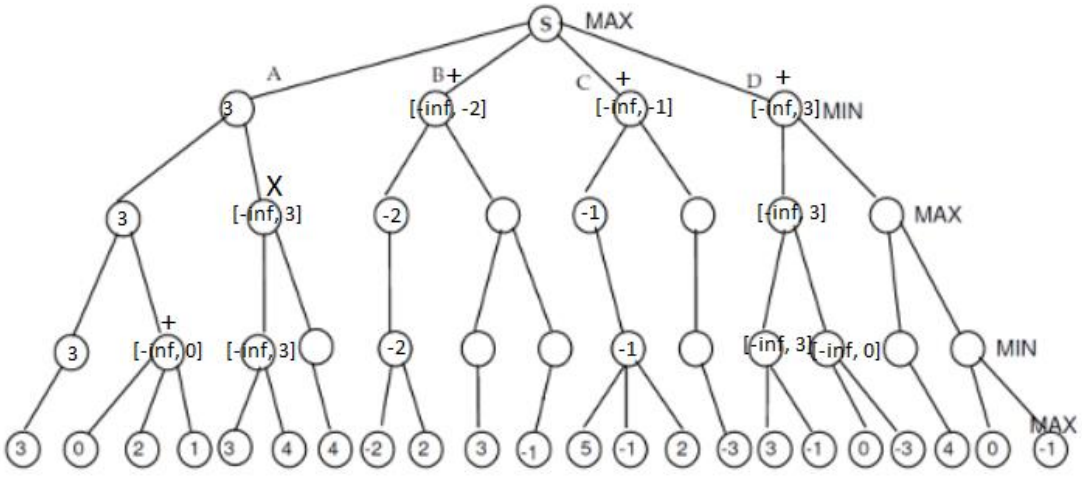
En el subárbol C vemos como en la primera rama obtenemos 1 y en la segunda rama el mejor valor posible es -1, por la que la podamos tambien.

En el subárbol D, vemos como el mejor valor posible es 0, por lo que tambien lo podamos. Obtenemos así finalmente el valor 3 como la solución, quedándonos con el valor del subárbol A que ya está garantizado a diferencia de B y D. Nos queda así el siguiente esquema:



Ejercicio 2.4

Ponemos aquí la imagen del árbol y sus podas y las explicamos a continuación:



Primero estudiamos el subárbol A, aquí, primeramente, vemos como subimos el 3 al tercer nivel, ya que, como el primer elemento del otro subárbol es 0 (0<3), solo podremos obtener el 0 o valores menores, que al pasar al tercer nivel siendo un Max siempre van a ser descartados. Realizamos así una poda por Max. Ahora, en el cuarto nivel del subárbol llevamos el 3 y el 3; este 3 lo obtenemos de ser el primer nodo del siguiente subárbol. Todos los valores que lleguen serán inferiores o iguales a 3, por lo que, en el mejor de los casos, el resultado sería el ya obtenido, podamos por Min.

En el subárbol B, cogemos el primer nodo que tiene valor -2 y al tratarse de un Min el segundo escalón, los resultados que obtengamos serán -2 o inferiores, al ser un valor inferior al 3 obtenido, no va a pasar al último escalón que es un Max, por lo que lo podamos.

En el subárbol C, obtenemos el primer nodo que es 5, al ser 5>3, seguimos ya que no nos permite podar. El siguiente valor es -1, que al ser inferior, nos garantiza que al pasar al segundo escalón, el valor que obtengamos será de menos infinito a -1, que al ser un valor menor que 3, podamos por Max.

En el subárbol D, el primer nodo es 3, así que los valores que obtendremos serán 3 o inferiores, por lo que podamos por Max en el penúltimo escalón al ser igual en el mejor de los casos al resultado ya obtenido.

Viendo el árbol ya estudiado, las ramas que se quedan sin explorar son las ramas que no tienen ningún número ni rango hacia arriba, al ser podadas por sus ramas izquierdas.